UNITEX

USER MANUAL

Université de Marne-la-Vallée

http://www-igm.univ-mlv.fr/~unitex
unitex@univ-mlv.fr

Sébastien Paumier - January 2003

English translation by Clemens Marschner, Johannes Stiehler, Friederike ... CIS,
Ludwig-Maximilians-Universität, Munich - Oct 2003
http://www.cis.uni-muenchen.de/

Introduction

Unitex is a collection of programs that can can handle texts in natural languages by using linguistic tools. These tools consist of electronic dictionaries, grammars and lexical grammar tables. These works were published for French by Maurice Gross of the Laboratoire d'Automatique Documentaire et Linguistique (LADL). They were extended to other languages through the RELEX laboratory network.

The electronic dictionaries contain the simple and composite words of each language associated with their lemmas and a series of grammatical (semantic and inflexional codes. The presence of these dictionaries is a major advantage in contrast to usual utilities for pattern search, since you can use the information that they contain for searching, and also describe large classes of words using very simple patterns.

The dictionaries are described in the DELA formalism and were worked out by linguists for different languages (French, English, Greek, Italian, Spanish, German, Thai, Korean, Polnish, Norvegian, Portugues, etc.)

The grammars are representations of linguistic phenomenons for recursive transition networks (RTN), a formalism close to finite state automata. Numerous studies have proved the adequacy of automata for linguistic problems, i.e. for morphology, syntax, and phonetic problems. The grammars created with Unitex take on this principle by using a formalism even more powerful than automata. The grammars are represented as graphs that the user can easily create and use.

The lexical grammar tables describe the properties of certain words. They were worked out for all simple words in French. They descriptions contain their syntactical proeprties.

Experience has shown that each word that has a quasi-unique behaviour, these tables allow for getting the grammar of every lexical element, (from which) the name of the lexical grammar (...) Unitex offers a way to construct the grammars from these tables.

Unitex is a motor with which you can exploit these linguistic ressources. Its technical characteristics are its portability, the modulartiity, the possibility to manage languages that use special writing styles (like Asian languages), and its openness, thanks to an open source distribution. Its linguistic characteristics are the ones that have motivated the elaboration of these resources: The precision, the completeness, and taking into account ??fignent?? problems, most notably those which cope with the ??recensement?? of composite words.

The first chapter describes installing and starting Unitex.

Chapter 2 presents the different steps for analyzing a text.

Chapter 3 describes the formalism of the DELA electronic dictionaries and the different operations that can be applied.

Chapters 4 and 5 present different means to make text searches more effective. Chater 4 describes in detail how the graph editor is used.

Chapter 6 is concerned with the different possible usage modes for grammars. The particularities of each grammar type are presented.

Chapter 7 introduces the automaton concept and describes the pecularities of this topic.

Chapter 8 comprehends an introduction to lexical-grammar tables, followed by a description of the method of constructing grammars basaed on these tables.

Chapter 9 describes in detail the different external programs that constitute Unitex.

Chapter 10 comprises a description of all file formats used in the system.

Chapter 1

Installation of Unitex

Unitex is a multi-platform system that runs on Windows as well as on Linux. This chapter describes how Unitex is installed and started on any of these systems. It also contains the procedures to add new languages and the deinstallation.

1.1 The Java runtime environment

Unitex comprises of a graphical interface written in Java and external programs written in C. This mixture of programming languages allows to get a fast and portable application that runs on different operating systems.

Before you can use the graphicsal interface you first have to install a runtime environment, usually called or ().

For the graphical mode, Unitex needs Java version 1.4 (or newer). If you have an older version of Java, Unitex will stop after you have chosen the working language.

You can download the virtual machine for your operating system for free from the Sun Microsystems web site at the following address: `http://java.sun.com`.

If you're working on Linux, or if you're using a Windows version with personal user accounts you have to ask your system administrator to install Java.

1.2 Installation on Windows

If you are the only user on your machine, you can perform the installation for yourself.

Decompress the file `unitex.zip` (You can download this file from the following address: `http://www-igm.univ-mlv.fr/~unitex`) into a directory `Unitex` that you should create, preferably within the `Program Files` folder.

After decompressing the file, the `Unitex` directory contains various subdirectories of which one is called `App`. This directory contains a file called `Unitex.jar`. This file is the Java executable that starts the graphical interface. You can double click on this icon to start the program. To facilitate the starting process, you may want to add a shortcut to the desktop.

1.3 Installation on Linux

In order to install Unitex on Linux, it is recommended to be a system administrator. Decompress the file `unitex.zip` to a directory named `Unitex`. Within the directory `Unitex/App`, start the shell script `make_exe` to compile the external programs. Limit the access rights to read and execute. After this you can create an alias in the following way:

```
alias unitex='cd /..../Unitex/App/ ; java -jar Unitex.jar'
```

1.4 First Start

If you work on Windows, the program will ask you to choose a personal work directory, which you can change later as well. To create a directory, click on the icon showing a file (see figure 1.3).

On Linux, the program will automatically create a `/unitex` directory in your `$HOME` directory. This directory allows you to save your personal works. For each language that you use, the program will copy a root directory of that language to your peronal work directory, except the dictionaries. You can also modify your copy of the files without the risk to damage the system files.

Welcome
Welcome spaumier!
To use Unitex, you must choose a private directory to store your data (that you can change later if you want).
Click on OK to choose your directory.
OK

Figure 1.1: First usage on Windows

Figure 1.2: First usage on Linux

Figure 1.3: Creating the personal work directory

1.5 Adding languages

There are different ways to add languages. If you want to add a language for all the users, you have to copy the corresponding directory to the `Unitex` system directory, for which you will need to have the access rights to this directory (maybe this means you need to ask your system administrator to do it). In return, if the language is only used by a single user, he can also copy the directory to his working directory. He can work with this language without this being exposed to other users.

1.6 Deinstallation

If you work on Windows or on Linux it is sufficient to delete the `Unitex` directory to clean your system from the program files. On Windows you may have to delete the shortcut to `Unitex.jar` if you have created one on your desktop. The same has to be done on Linux if you have created an alias.

Chapter 2

Loading texts

One of the profound functions of Unitex is the search for expressions within a text. For this matter, the texts have to be undertaken a set of preprocessing steps that perform a normalization of non-ambigue forms and split the texts up in phrases. Once these operations are done, the electronic dictionaries are applied on the texts. After this you can search more effectively in the texts by using these grammars.

This chapter describes the different steps for text preprocessing.

2.1 Selecting the language

When starting Unitex, the program asks you to choose the language in which you want to work (see figure 2.1). The languages displayed are the ones that are present in the system directory `Unitex` and those that are probably installed in your personal working directory. If you use a language for the first time, Unitex copies the system directory of this language to your personal directory, except the dictionaries.

Choosing the language allows Unitex to find certain files, for example the alphabet file. You can change the language all the time by choosing "Change Language..." in the "Text" menu. If you change the language, the program will close all windows relative to the current text, if there are any. The active language is indicated in the title bar of the graphical interface.

2.2 Text formats

Unitex works with Unicode texts. Unicode is a standard that describes a universal character code. Each character is given a unique number which allows to represent texts without having to account for the proper codes on different machines and/or operating systems. Unitex uses a two-byte representation of the Unicode 3.0 standard, called Unicode Little-Endian (for more details, see `http://www.unicode.org`).

The texts that come with Unitex are already in Unicode format. For testing whether a text is in Unicode or not, the simplest way is to try to open it with Unitex. An error

Figure 2.1: Language selection when starting Unitex

message will tell you if the text is not in Unicode. On Linux, you can use the command `less file_name`. If the text shows normally it is not in Unicode.

In order to get the text to the right format, the easiest way is to process the text and save the document in the "Unicode text" format.

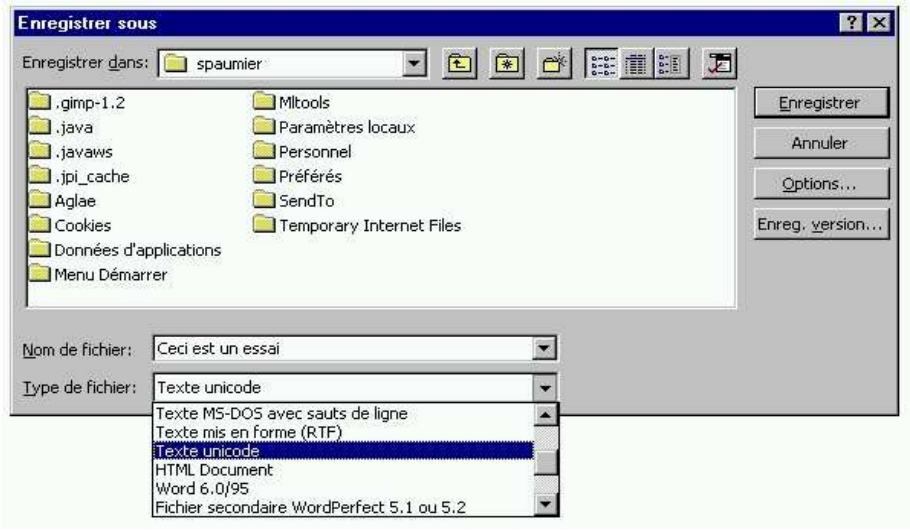

Figure 2.2: Save in Unicode format in Microsoft Word

By default, the code format on a PC is always Unicode Little-Endian. If your text file is not formatted (without layout or color information) and was created on Windows, you can also use the program `Asc2Uni` on the command line.

This program converts the text files from ASCII or Windows ANSI

(see chapter 9).

The resulting texts don't contain any formatting information (layout, colors, etc.) and are ready to be used with Unitex

2.3 Opening a text

Unitext offers to open two types of text files.

The files with the extension `.snt` are text files preprocessed by Unitex which are ready to be manipulated by the system functions. The files ending with `.txt` are raw files.

To use a text, you have to start by opening the `.txt` file by clicking on "Open..." in the "Text" menu.

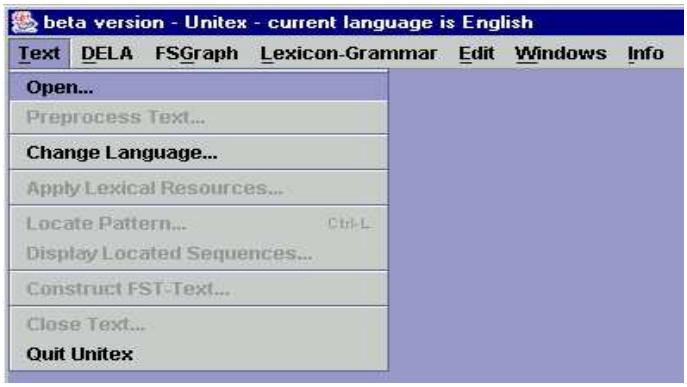

Figure 2.3: Text Menu

Choose the file type "Raw Unicode Texts" and select your text:

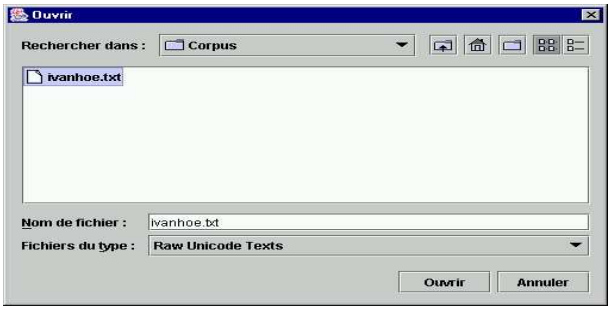

Figure 2.4: Opening a Unicode text

Files larger than 5 MBytes are not displayed. The message "`This file is too large to be displayed. Use a word processor to view it.`" is displayed in the window. This behavior applies to all open text files (the list of lexical units, dictionaries, etc.).

2.4 Text preprocessing

After a text is selected, Unitex offers you to preprocess it. The text preprocessing consists of performing the following operations: Normalization of seperators, discovery of lexical

units, and the use of dictionaries.

If you refuse the preprocessing, the text will nevertheless be normalized and lexical units looked up, since these operations are necessary for Unitex to work. It is always possible to carry out the preprocessing later by clicking on "Preprocess Text..." in the "Text" menu.

If you accept to preprocess the text, Unitex proposes to parameterize it in the window shown in figure 2.5.

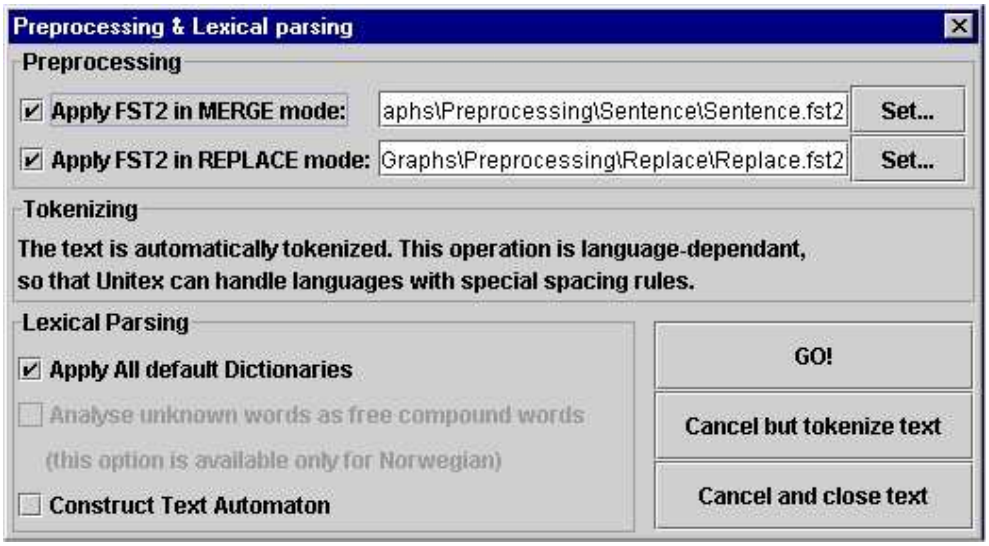

Figure 2.5: Preprocessing Window

The option "Apply FST2 in MERGE mode" helps cutting up the text in phrases. The option "Apply FST2 in REPLACE mode" is used to make replacements in the text more effective, especially in the normalization of non-ambiguous forms. With the option "Apply All default Dictionaries" you can apply dictionaries in the DELA format (Dictionnaires Electroniques du LADL). The option "Analyse unknown words as free compound words" is used in Norvegian for correctly analyzing free-formed composite words ??pour analyser correctement les mots composés libres formés par soudure de mots simples??.

Finally, the option "Construct Text Automaton" is used to build the text automaton. This option is deactivated by default, because it consumes a large amount of memory and disk space if the text is too large. The construction of the text automaton is described in chapter 7.

NOTE: If you click on "Cancel but tokenize text", the program will carry out the normalization of separators and will find lexical units. Click on "Cancel and close text" to abandon the operation completely.

2.4.1 Normalisation of Separators

Usual separators are the space, the tab and the newline characters. There can be several separators following upon another, but since this isn't useful for linguistic analyses, separators are normalized according to the following rules:

- At first separators that contain at least one line break are replaced by a single line break
- all other sequences of separators are replaced by a single space.

The distinction between space and new line is held up at this point because the presence of line breaks may have an influence on the process of seperating the text into phrases. The result of the normalization of a text named `my_text.txt` is a file in the same directory like the `.txt` file and is named `my_text.snt`.

NOTE: When the text is preprocessed using the graphical interface, a file named `my_text_snt` is created immediately after the normalization. This repository, called text repository, contains all the given relatives of this text.

2.4.2 Phrase Detection

The phrase detection is an important preprocessing step since it allows for creating units for linguistic processing. This detection is used by the text automaton construction program. In contrast to what you might think, detecting phrase boundaries is not a trivial problem. Consider the following text:

The family has urgently called Dr. Martin.

The full stop that follows *Dr* is followed by a word beginning with a capital letter. Thus it may be considered as the end of the phrase, which will be wrong. To avoid these kind of problems caused by the ambiguous use of punctuation, grammars that describe the different contexts or that can show the phrase endings are used. Figure 2.6 shows an example grammar for phrase detection..

When a path of the grammar recognizes a sequence in the text and when this path produces the phrase separator symbol `{S}` , this symbol is inserted into the text.

The path shown at the top of figure 2.6 recognizes the sequence made up of a question mark and a word beginning with a capital letter and inserts the symbol `{S}` between the question mark and the following word. The following text:

What time is it? Eight o' clock.

will be converted to:

What time is it ?{S} Eight o' clock.

A detection grammar might change the following special symbols:

- <E> : empty word, or epsilon. Recognizes an empty sequence;
- <MOT> : recognizes sequences of letters;
- <MIN> : recognizes sequences of letters in lower case;
- <MAJ> : recognizes sequences of letters in upper case

- <PRE> : recognizes sequences of letters that begin with an uppercase word
- <NB> : recognizes sequences of numbers (1234 is recognizes but not 1 234);
- <PNC> : recognizes the punctuation symbols ; , ! ? : and the inversed exclamation points and question marks in Spanish and some Asian punctuation letters
- <^> : recognizes a new line;
- # : recognizes a space.

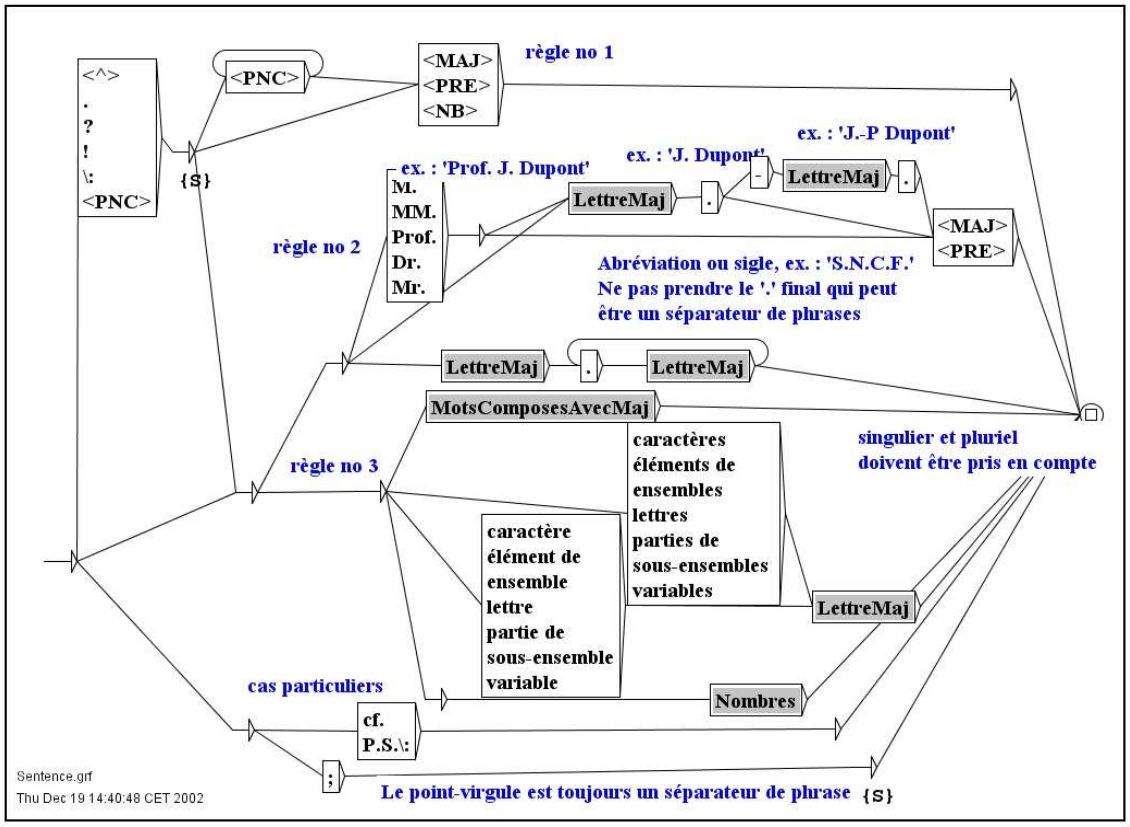

Figure 2.6: Phrase detection grammar for French

By default, the space is optional beween two boxes [[?]]. If you wont to prohibit the presence of this separator you have to use a special separator, #. Lower and upper case letters are defined by a file alphabet

(see chapter **??**). For more details on grammars, see chapter 5. The grammar used here is named `Sentence.fst2` and can be found in the following repository:

```
/(user home directory)/(language)/Graphs/Preprocessing/Sentence
```

Applicating this grammar to a text is done through the `Fst2Txt` program in MERGE mode. This signals that the output produced by the grammar, in this case the symbol `{S}`, is inserted into the text. This program takes a `.snt` file and modifies it.

2.4.3 Normalization of non-ambigue forms

Certain forms present in texts can be normalized (for example the sequence in French, "*l'on*" is equivalent to the form "*on*". A user may want to replace these forms more effectively according to his needs. However, you have to be careful that the normalized forms are non ambigue and that the removal of ambiguities doesn't have consequences on the applications sought.

[[Toutefois, il faut faire attention à ce que les formes normalisées soient non ambiguës, ou à ce que la disparition de l'ambiguïté soit sans conséquence pour l'application recherchée.]]

If you decide to replace the form *audit* by *à le-dit*, the phrase:

La cour a procédé à un audit des comptes de cette société.

will be replaced by the following incorrect sentence:

La cour a procédé à un à le-dit des comptes de cette société.

Thus you have to be very careful when you change the normalization grammar.

You have to pay attention to spaces as well. For example, if you replace *c'* by *ce* not followed by a space, the phrase:

Est-ce que c'était toi?

will be replaced by the following incorrect sentence:

Est-ce que ceétait toi?

The accepted symbols for normalization grammars are the same as the ones allowed for phrase detection. The grammar used is named `Replace.fst2` and can be found int the following directory:

```
/(home directory)/(active language)/Graphs/Preprocessing/Replace
```

As with phrase detection, this grammar is applied using the program `Fst2Txt`, but this time in REPLACE mode,, which signifies that the [[entrées]] recognised by the grammar are replaced by the sequences produced by them. In figure **??** you can see a grammar that resolves [[resoud]] some [[élisions]] in French.

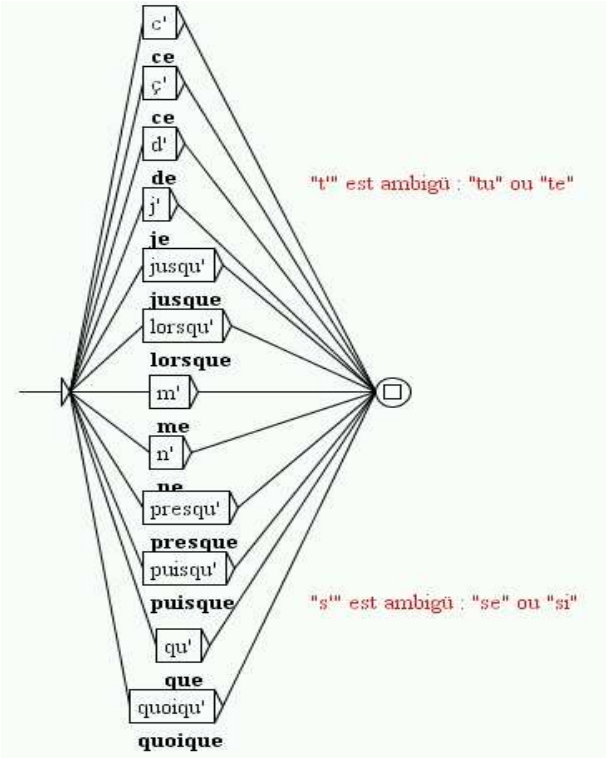

Figure 2.7: normalization grammar for some [[élisions]] in French

2.4.4 Splitting a text into lexical units

Some languages, in particular Asian languages, use separators that are different than the ones used in the western world. Spaces can be forbidden, optional, or mandatory. In order to better cope with these particularities, the way Unitex splits texts is language dependent.

Thus, a language like French is treated the following way:

A lexical unit can be:

- the phrase separator `{S}`;
- a lexical name [[étiquette]]`{aujourd'hui,.ADV}`;
- a sequence of letters [[suite contiguë de lettres]] (the letters are defined in the language alphabet file);
- a non-word character; If it is a new line, it is replaced by a space.

For other languages, splitting is done character by character, except for the phrase separator `{S}` and lexical names.

The basic splitting guarantees Unitex to work, but limits the optimization of search for patterns.

Regardless of the mechanism used, the new lines in a text are replaced by spaces.

The split is done by the `Tokenize`. program. This program creates several files that are saved in the text directory:

- `tokens.txt` contains the list of lexical units in the order in which they are found in the text;
- `text.cod` contains the entity table; every entity corresponds to the index of of a lexical unit in the file `tokens.txt`;
- `tok_by_freq.txt` contains the list of lexical units sorted by frequency;
- `tok_by_alph.txt` contains the list of lexical units in alphabetical order;
- `stats.n` contains some statistics about the text.

Splitting up the text:

Un sou c'est un sou.

returns the following list of lexical units: *Un* ESPACE *sou c ' est un .*

You may remark that the rules are case sensitive (*Un* and *un* are two distinct entities), but that each entity is encoded only once. For enumerating the numbers 0 to 7, the text can be representet by a sequence of entities as described in the following table:

Indice	0	1	2	1	3	4	5	1	6	1	2	7
Corresponding lexical unit	*Un*		*sou*		*c*	*'*	*est*		*un*		*sou*	.

Table 2.1: Representing the text *Un sou c'est un sou.*

For more details, see chapter **??**.

2.4.5 Applying dictionaries

Applying dictionaries consists of building the subset of dictionaries only consisting of forms present in a text. So, the result of applying a French dictionary to the text *Igor mange une pomme de terre* produces a dictionariy of the following simple words:

```
de,.DET+z1
de,.PREP+z1
de,.XI+z1
mange,manger.V+z1:P1s:P3s:S1s:S3s:Y2s
pomme,.A+z1:ms:fs:mp:fp
```

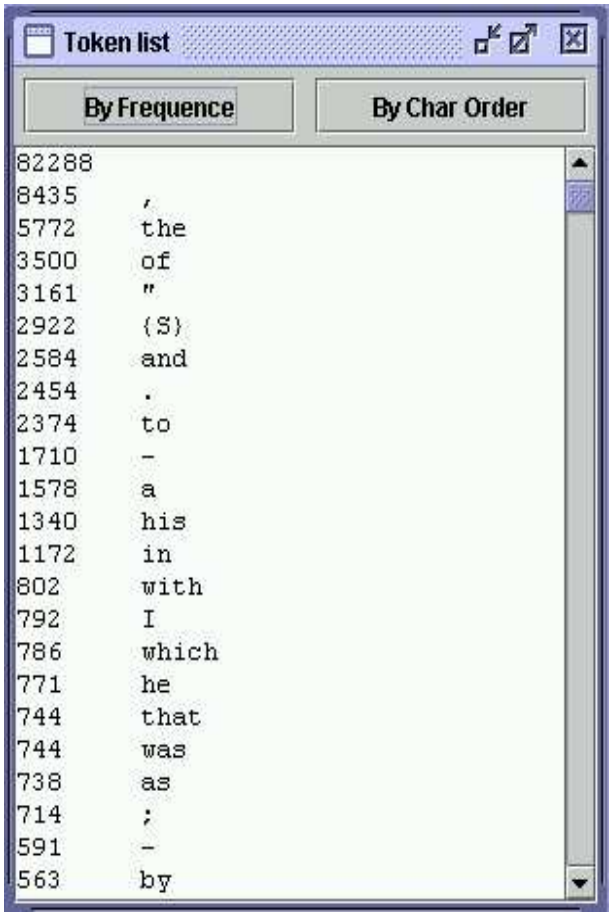

Figure 2.8: Lexical units in an english text sorted by frequency

```
pomme,.N+z1:fs
pomme,pommer.V+z3:P1s:P3s:S1s:S3s:Y2s
terre,.N+z1:fs
terre,terrer.V+z1:P1s:P3s:S1s:S3s:Y2s
une,.N+z1:fs
une,un.DET+z1:fs
```

as well as a dictionary of composite words consisting of a single entry:

```
pomme de terre,.N+z1:fs
```

Since the sequence *Igor* is neither a simple French word, nor a part of a composite word, it was considered like an unknown word. The application of dictionaries is done through the program `Dico`. The three files produced (`dlf` for simple words, `dlc` for composite words and `err` for unknown words) are placed in the text folder. Text dictionaries are called the `dlf` and `dlc` files.

As soon as the application of dictionaries is finished, Unitex presents the sorted list of simple, composite and unknown words found in a Window. Figure 2.9 shows the result for a French text.

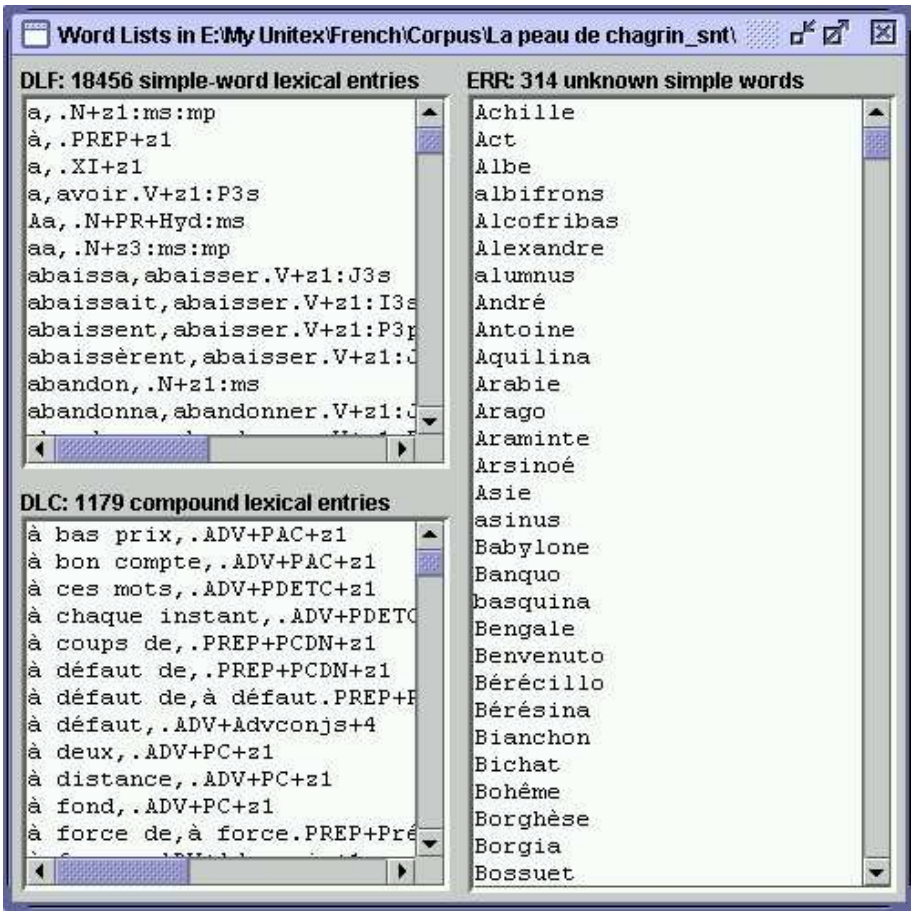

Figure 2.9: Result after applying dictionaries on a French text

It is also possible to apply dictionaries without preprocessing the text. In order to do this, you have to click on "Apply Lexical Resources..." in the "Text" menu. Unitex then opens a window (see figure 2.10) that allows for choosig te list of dictioaries to apply.

The list "User resources" summarizes all compressed dictionaries present in the `(current language)/D` of the user. The dictionaries installed in the system are listed in the box named "System resources". Use the combination<Ctrl+click> to select multiple dictionaries. The button "Set Default" allows you to define the current dictionary selection as a default. This default selection will then be used during preprocessing if you activate the option "Apply All default Dictionaries".

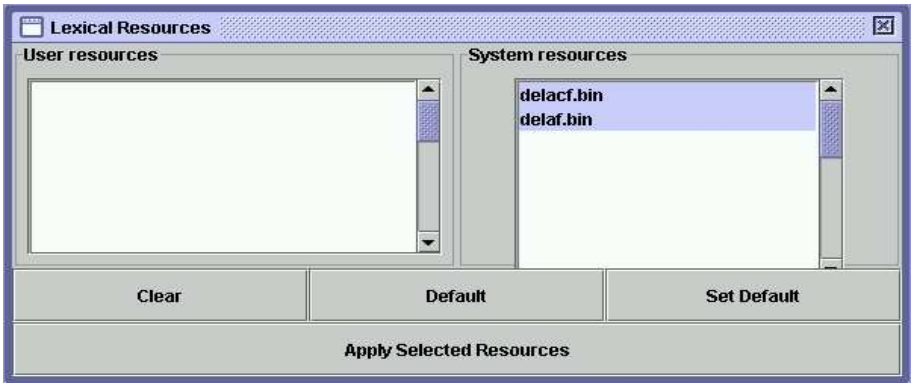

Figure 2.10: Parameterizing the application of dictionaries

2.4.6 Analysis of composite words in Norvegian

In Norvegian, it is possible to form free composite words by knitting together their elements. For example, the word *aftenblad* meaning *evening journal* is obtained by combining the words *aften* (*evening*) et *blad* (*journal*). The program `PolyLex` searches the list of unknown words after the application of dictionaries and tries to treat each of these words as a composite word. If a word can be resolved to at least one composite word, it is deleted from the list of unknown words and the lineds produced for this word are appended to the text dictionary of simple words

Chapter 3

Dictionaries

3.1 The DELA Dictionaries

The electronic dictionaries used by Unitex apply the DELA syntax (Dictionnaires Electroniques du LADL, LADL electronic dictionaries). This syntax describes simple and composite lexical entries of a language with their grammatical, semantic and inflectional information. We distinguish between two kinds of electronic dictionaries. The one that is used most often is the inflected form dictionary DELAF (DELA de formes Fléchies, DELA of inflected forms) or DELACF (DELA de formes Composées Fléchies, DELA of composite inflected forms) if it is concerned with composite forms. The second type is a dictionary of non inflected forms called DELAS (DELA de formes Simples, Simple forms DELA) or DELAC (DELA de formes Composées, composite forms DELA).

Unitex programs don't make a distinction between simple and composite form dictionaries. We will use the terms DELAF and DELAS to distinguish between the two kinds of dictionaries whose entries are simple, composite, or mixed forms.

3.1.1 The DELAF Format

Entry syntax

An entry of a DELAF is a line of text terminated by a newline that conforms to the following syntax:

```
mercantiles,mercantile.A+z1:mp:fp/this is an example
```

The different elements of this line are:

- `mercantiles` is the inflected form of the entry. ; it is mandatory;

- `mercantile` is the canonical form of the entry. For nouns and adjectives, it is usually the masculine singular form; for verbs, it is the infinitive. This information may be be left out like in the following example:

```
boîte à merveilles,.N+z1:fs
```

This signifies that the canonical form is the same as the inflected form. The canonical form is separated from the inflected form by a comma.

- `A+z1` is the sequence of grammatical and semantic information. In our example, `A` designates an adjective, and `z1` shows that it is a common word (see table3.2).

  Each entry can carry at least one grammatical and semantic entry, separated from the canonical form by a full stop. If there are more codes, these may be separated by the `+;` character.

- `:mp:fp` is a sequence of inflectional information.

  This information describes the genus, numerus, the tense and the the conjugation modes, the declinations for the ??langues à cas??, etc. This information is optional. A inflectional code is made up of one or more characters that represent one information each. Inflectional codes have to be separated by the `:` character.

  In our example, `m` signifies masculin, `p` plural and `f` feminin (see table 3.3). The `:` character is interpreted in OR semantics. Thus `:mp:fp` means "plural masculine" ou "plural feminin". Since each character represents one information, it is not necessary to use the same character more than once. This way encoding the past participle using the code `:PP` would be exactly equivalent to using `:P` alone;

- `/this is an example` is a comment. Comments are optional and may be introduced by the `/` character. These comments are left out when the dictionaries are compressed.

IMPORTANT REMARK: It is possible to use the full stop and the comma within a dictionary entry. In order to do this they have to be escaped using the `\:` character.

```
3\,1415,PI.NOMBRE
United Nations Organization,U\.N\.O\..SIGLE
```

ATTENTION: Each character is taken into account within a dictionary line. For example, if you insert spaces, they are considered to be a part of the information. In the following line:

```
gît,gésir.V+z1:P3s /voir ci-gît
```

the space that precedes the `/` character will be considered to be one of the 4 inflectional codes `P`, `3`, `s` et d'un espace.

Composite words with space or dash

Certain composite words like *grand-mère* can be written using spaces or dashes. In order to avoid having to double the entries, it is possible to use the = character. At the time when the dictionary is compressed, the `Compress` verifies for each line if the inflected or canonical form contains a non-escaped = character. If this is the case, the programm replaces this by two entries: The one with the = character is replaced by a space, and one where it is replaced by a dash. Thus, the following entry:

```
grand=mères,grand=mère.N:fp
```

is replaced by the following entries:

```
grand mères,grand mère.N:fp
grand-mères,grand-mère.N:fp
```

NOTE: If you want to keep an entry that includes the = character, escape it using \ like in the following example:

```
E\=mc2,.FORMULE
```

This replacement is done when the dictionary is compressed. In the compressed dictionary, the escaped = characters are replaced by simple =. As such, if a dictionary containing the following lines is compressed:

```
E\=mc2,.FORMULE
grand=mère,.N:fs
```

and if the dictionary is applied to the following text:

Ma grand-mère m'a expliqué la formule E=mc2.

you will get the following lines in the dictionary of composite words of the text:

```
E=mc2,.FORMULE
grand-mère,.N:fs
```

??Factorisation?? of entries

Several entries containing the same inflectional and canonical forms can be regrouped to a single one if they have the same grammatical and semantic codes. This allows among others for regrouping identical conjugation for a verb:

```
glace,glacer.V+z1:P1s:P3s:S1s:S3s:Y2s
```

s

If the grammatical and semantic information differs, you have to create distinct entries:

```
glace,.N+z1:fs
glace,glacer.V+z1:P1s:P3s:S1s:S3s:Y2s
```

Certain antries having the same grammatical and semantic entries can have different senses, like it is the case for the word *poêle* that describes a stove or a net in the masculine sense and a kitchen instrument in the feminine sense. You can thus distinguish the entries in the following case:

```
poêle,.N+z1:fs/ poêle à frire
poêle,.N+z1:ms/ voile, linceul; appareil de chauffage
```

NOTE: In practice this distinction has the single consequence that the number of entries in the dictionary rises.

In the different programs that make up Unitex these entries are reduced to

```
poêle,.N+z1:fs:ms
```

Whether this distinction is made is thus left to the people that maintain the dictionaries.

3.1.2 The DELAS Format

The DELAS format is very similar to the one used in DELAF. The only difference is that there is only one canonical form followed by grammatical and/or semantic codes. The canonical form is separated from the different codes by a comma. See this example:

```
cheval,N4+Anl
```

The first grammatical or semantic code will be interpreted by the ??programme de flection?? as the name of the grammar used to inflect the entry. The entry of the example above indicates that the word *cheval* has to be inflected using the grammar named `N4`. It is possible to add flectional codes to the entries, but the nature of the inflection operation limits the usefulness of this possibility. For more details see below in section **??**.

3.1.3 Dictionary Contents

The dictionaries shipped with Unitex contain descriptions of simple and composite words. These descriptions indicate the grammatical category of each entry, optionally their flectional codes, and diverse semantic information. The following tables give an overview of the different codes used in the dictionaries shipped with Unitex. These codes are the same for almost all languages, though some of them are special for certain languages (*i.e.* ??marque du neutre??, etc.).

Code	Description	Examples
A	adjectif	fabuleux
ADV	adverbe	réellement, à la longue
CONJC	??conjonction de coordination??	mais
CONJS	??conjonction de subordination??	puisque, à moins que
DET	determiner	ses, trente-six
INTJ	??interjection??	adieu, mille millions de mille sabords
N	??nom??	prairie, vie sociale
PREP	preposition	sans, à la lumière de
PRO	pronom	tu, elle-même
V	verb	continuer, copier-coller

Table 3.1: Frequent grammatical codes

Code	Description	Example
z1	??langage courant??	blague
z2	??langage spécialisé??	sépulcre
z3	??langage très spécialisé??	houer
Abst	??abstrait??	bon goût
Anl	animal	cheval de race
AnlColl	??animal collectif??	troupeau
Conc	??concret??	abbaye
ConcColl	??concret collectif??	décombres
Hum	human	diplomate
HumColl	??humain collectif??	vieille garde
t	transitive verb	foudroyer
i	intransitive verb	fraterniser
en	??particule pré-verbale (PPV) obligatoire??	en imposer
se	pronominal verb	se marier
ne	??verbe à négation obligatoire??	ne pas cesser de

Table 3.2: Some semantic codes

NOTE: The time descriptions in table 3.3 correspond to French. Nontheless, the majority of these definitions can be found in other languages (infinitive, present, ??participe passé??, etc.).

Except for a common base in the majority of languages, the dictionaries contain encoding particularities that are special for each language. Thus, as the declination codes vary a lot between different languages, they are not described here. For a complete decription of all codes used within a dictionary, we recommend to contact the author of a dictionary directly.

However, these codes are not restrictive. Every user can introduce codes himself, and can

Code	Description
m	masculine
f	feminin
n	neuter
s	singular
p	plural
1, 2, 3	1st, 2nd, 3rd person
P	present indicative
I	??imparfait de l'indicatif??
S	??présent du subjonctif??
T	??imparfait du subjonctif??
Y	present imperative
C	??présent du conditionnel??
J	??passé simple??
W	infinitive
G	??participe présent??
K	??participe passé??
F	future

Table 3.3: Common flectional codes

create clean dictionaries. For example, one could in a pedagogical goal introduce markers that indicate French ??"faux-amis"??:

```
bless,.V+faux-ami/bénir
cask,.N+faux-ami/tonneau
journey,.N+faux-ami/voyage
```

It is equally possible to use dictionaries to add extra information. Thus, you can use the inflected form of an entry to describe an abbreviation and the canonical form to provide the complete form:

```
ADN,Acide DésoxyriboNucléique.SIGLE
LADL,Laboratoire d'Automatique Documentaire et Linguistique.SIGLE
SAV,Service Après-Vente.SIGLE
```

3.2 Verfication of the dictionary format

As dictionaries are very important, it becomes tiresome to verify them instantly. Unitex contains the `CheckDic` that automatically verifies the DELAF and DELAS dictionaries.

This program verifies the syntax of the entries. For each malformed entry, the program outputs the line number, the contents of this line and the type of error. The results are saved in the file `CHECK_DIC.TXT` which is displayed when the verification ends. In addition to possible error messages, the file contains the list of all characters used in the flexional and

canonical forms, the list of grammatical and semantic codes, and the list of flexional forms used. The character list makes it possible to verify that the characters used in the dictionary are consistent with those in the file alphabet of the language. Each character is followed by its value in hexadecimal notation.

These code lists can be used to verify that there are no encoding errors ??faute de frappe?? in the codes of the dictionary.

The program works with non-compressed dictionaries, i.e. the files in text form. The general convention used is to use the `.dic` extension for these dictionaries. In order to verify the format of a dictionary, you first have to open it by choosing "Open..." in the "DELA" menu

Figure 3.1: "DELA" Menu

Let's load the dictionary as in figure 3.2:

Figure 3.2: Dictionary example

In order to start the automatic verification, click on "Check Format..." in the "DELA" menu. A window like in figure 3.3 is opened:

In this window you choose the dictionary type you want to verify. The results of this verification in figure 3.2. are displayed in figure 3.4.

The first error is caused by a missing full stop. The second, that no comma was found after the end of an inflected form. The third error indicates that the program hasn't found any grammatical or semantic codes.

Figure 3.3: Automatic verification of a dictionary

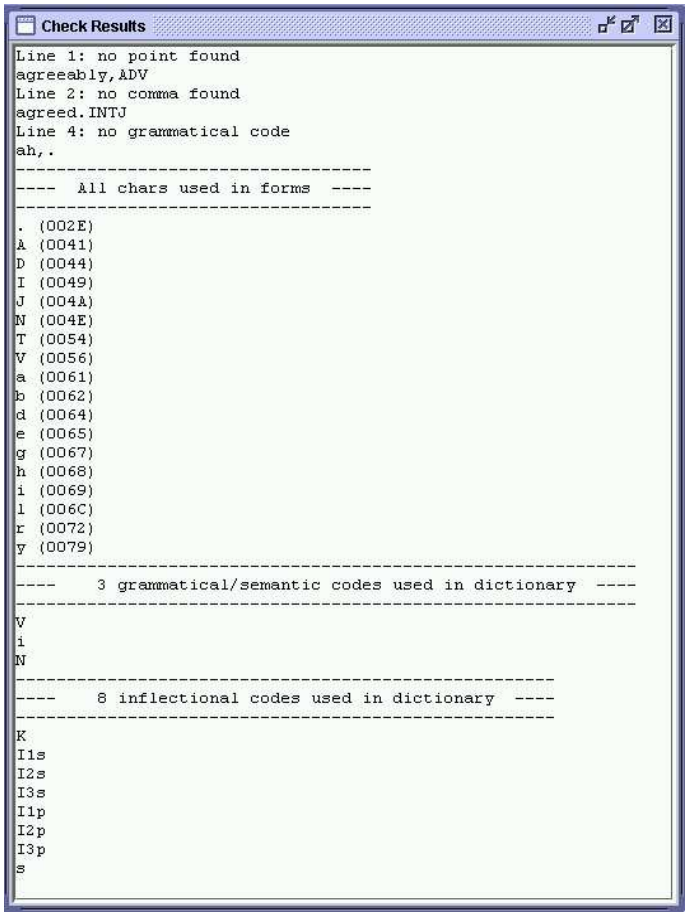

Figure 3.4: Results of the automatic verification

3.3 Sorting

Unitex uses dictionaries without paying attention to the entry order. But in order to get a better presentation it is often better to sort the dictionaries. The sort operation is performed along different criterias, beginning with the language of the text to sort. Thus, sorting a Thaï dictionary will result in a different alphabetic order, since Unitex uses a special sort mechanism especially developed for the Thaï language (see chapter 9).

For European languages, sorting is generally done in lexicographical order, although there are a couple of variants. Certain languages like the French consider different characters to be equivalent. For example, the difference between `e` an `é` is ignored in order to compare words like `manger` and `mangés`, and let the contexts `r` and `s` decide the order. The distinction is not made unless the contexts are identical, which is the case if `pêche` and `pèche` is compared.

In order to cope with this phenomenon, the sort program `SortTxt` uses a file that contains equivalent characters. This file is named `Alphabet_sort.txt` and can be found in the active language dictionary of the user. The following are the first lines for French:

```
AÀÂÄaàâä
Bb
CÇcç
Dd
EÉÈÊËeéèêë
```

Characters within one line are considered equivalent if the context allows it. Whenever two equivalent characters are found, they are sorted in the order as they appear in the line, from left to right. You can see in the example above that no difference is made between lower case and uper case letters, and that accents like the cédille are ignored.

In order to sort a dictionary, open it, then click on "Sort Dictionary" in the "DELA" menu. By default, the program always tries to use the `Alphabet_sort.txt` file. If this file is not found, sorting is done according to the Unicode encoding. By modifying this file, you can specify your sorting preferences.

Remark: after applying a dictionary to a text, the files `dlf`, `dlc` and `err` are automatically sorted with this program.

3.4 Automatic flexion

As described in section 3.1.2, a line in a DELAS is comprised of a canonical form and a sequence of grammatical or semantic codes:

```
bocal,N4+Conc
cheval,N4+Anl
local,N4
```

The first code is interpreted as the name of the grammar used to flexion the canonical form. These flexional grammars have to be compiled (see chapter 5). In the example above, all entries will be flexioned by a grammar named `N4`.

In order to start a flexion, click on "Inflect..." in the "DELA" menu. The window in figure 3.5 allows for specifying the directory in which flexional grammars are found. By default, the subdirectory `Inflection` of the directory of the current language is used.

Figure 3.5: Configuration of automatic flexion

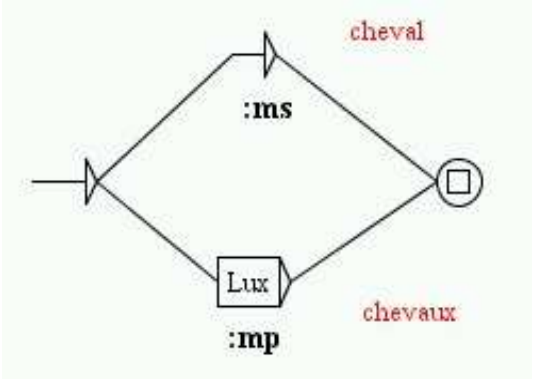

Figure 3.6: Flexional grammar`N4`

Figure 3.6 displays an example of a flexional grammar:

The paths describe the suffixes to add or to remove to get to a flexioned form from a canonical form, and the exits (text in bold under the boxes) are the flexional forms to add to a dictionary entry.

In our example, two paths are possible. The first doesn't modify the canonical form and adds the flexional code `:ms`. The second deletes a letter due to the `L` operator, then adds the `ux` suffix and adds the flexional code `:mp`. Two oeprators are possible: :

- `L` (left) remove a letter from the entry
- `R` (right) restore a letter to the entry. In French, many verbs of the first group are conjugated in the third person form of the present singular by removing the `r` of the infinitive and change the 4th lettre from the end to `è`: `peler` → `pèle`, `acheter` → `achète`, `gérer` → `gère`, etc. Instead of describing a flexional suffix for each verb (`LLLLèle`, `LLLLète` et `LLLLère`), the `R` operator can be used to describe it in one step: `LLLLèRR`.

The inflexion program, `Inflect` traverses all paths of the flexional grammar and tries all possible forms. In order to avoid having to replace the names of flexional grammars by the real grammatical codes in the dictionary used, the program replaces these names by

the longest prefixes made of letters. Thus, `N4` is replaced by `N`. By choosing the flexional grammar names carefully, you can make a dictionary ready to use directly.

Let's have a look at the dictionary we get after the DELAS flexion in our exapmle:

Figure 3.7: Result of automatic flexion

3.5 Compression

Unitex apples compressed dictionaries to the text. The compression reduces the size of the dictionaries and speeds up the lookup.

This operation is done by the `Compress` program. . This program takes a dictionary in text form as input (for example `my_dico.dic`) and produces two files:

- `my_dico.bin` contains the minimal automaton of the flexional formes of the dictionaries;
- `my_dico.inf` contains the codes that allow for reconstructing the original dicionary from the flexional forms in the `my_dico.bin` file.

The minimal automaton in the `my_dico.bin` file is a represenation of flexional forms in which all common prefixes and suffixes are factorized. For example, the minimal automaton of the words `me`, `te`, `se`, `ma`, `ta` et `sa` can be represented by the graph in figure 3.8.

For compressing a dictionary, open it and click on "Compress into FST" in the "DELA" menu. The compression is independent of the language and of the content of the dictionary. The messages produced by the program are displayed in a window that is not closed automatically. Figure 3.9 shows the result of the compression of a dictionary of simple words.

??À titre indicatif, les taux de compression?? The resulting files are compressed to about 95% for dictionaries containing simple words and 50% for those with composite words.

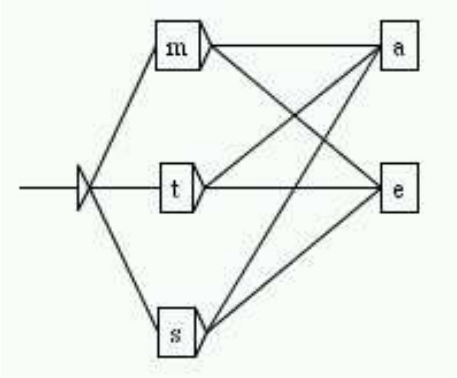

Figure 3.8: Represenation of an example of a minimal automaton

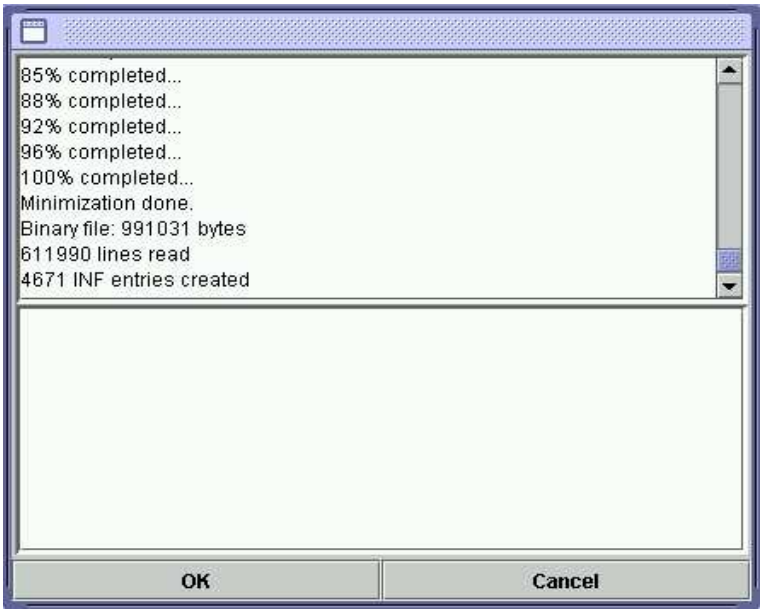

Figure 3.9: Results of a compression

3.6 Applying dictionaries

Dictionaries can be applied after pre-treatment, explicitly by clicking on "Apply Lexical Resources" in the "Text" menu (see section 3.6).

We will now describe in detail the rules for applying dictionaries.

3.6.1 Priorities

The priority rule is the following: if a word in a text is found in a dictionary, this word will not be taken into account by dictionaries with lower priority.

This allows for eliminating certain ambiguities when applying dictionaries. For example, the word *par* has a nominal interpretation in the golf domain. If you want to cope with this usage, it is sufficient to create a filter dictionary containing only the entry `par,.PREP` and to apply this with highest priority. This way, even if dictionaries of simple words contain a different entry, this will be ignored due to the priority rule.

There are three priority levels. The dictionaries whose names without extension end with `-` have the highest priority; those that end with `+` have the lowest one. All other dictionaries are applied with medium priority. The application order of different dictionaries having the same priority is not defined. On the command line, the command

```
Dico ex.snt alph.txt Pays+.bin Villes-.bin Fleuves.bin Regions-.bin
```

will apply the dictionaries in the following order (`ex.snt` is the text on which the dictionaries are applied, and `alph.txt` is the alphabet file used):

1. `Villes-.bin`
2. `Regions-.bin`
3. `Fleuves.bin`
4. `Pays+.bin`

3.6.2 Application rules for dictionaries

Besides the priority rule, the application of dictionaries is done while respecting upper and lowercase letters and spaces. The upper case rule is like follows:

- if there is an upper case letter in the dictionary, then an upper case letter has to be in the text;
- if a lower case letter is in the dictionary, there can be either an upper or lower case letter in the textdans le texte.

Thus, the entry `pierre,.N:fs` will match the words `pierre`, `Pierre` et `PIERRE`, while `Pierre,.N+Prénom` only recognizes `Pierre` and `PIERRE`. Lower and upper case letters are defined in the alphabet file passed to the `Dico`. as a parameter.

Respecting white space is a very simple rule: For each sequence in the text to be recognized by a dictionary entry, it has to have exactly the same number of spaces. For example, if the dictionary contains `aujourd'hui,.ADV`, the sequence `Aujourd' hui` will not be recognized because of the space that follows the apostrophe.

Chapter 4

Search for regular expressions

In this chapter we will see how to search for simple patterns in a text by using regular expressions.

4.1 Definition

The goal of this chapter is not to give an introduction on formal languages but to show how to use regular expressions in Unitex in order to search for simple topics. Readers who are interested in a more formal presentation can consult the many works that treat regular expression patterns.

A regular expression can be:

- a lexical unit (`livre`) or a topic (`<manger.V>`);Motive
- the concatenation of two regular expressions (`je  mange`);
- the union of two regular expressions (`Pierre+Paul`);
- the Kleene star of a regular expression (`très*`).

4.2 Lexical units

In a regular expression a lexical unit is a sequence of letters. The symbols point, plus, star, less than as well as the opening and closing brackets have a special meaning. It is therefore necessary to escape them with the character `\` if you want to search for them. These are some examples of valid lexical units:

```
chat
O\.N\.U\.
\(1984\)
{S}
```

By default Untiex accepts lower-case patterns and also finds upper-case words. It is possibe to enforce case-sensitive matching using quotation marks. Thus, `''pierre''` recognizes only the form `pierre` and not `Pierre` or `PIERRE`.

NOTE: in order to make a space obligatory, you have to enclose it in quotation marks.

4.3 Patterns

4.3.1 Special symbols

There are two kinds of patterns. The fist category contains all symbols that have been introduced in section 2.4.2 except for the symbol `<^>`, which matches a line feed. Since all line feeds have been replaced by spaces this symbol cannot longer be useful when searching for patterns. These symbols – called *meta-symbols* are the following:

- <E> : the empty word or epsilon. Matches the empty string;
- <MOT> : matches any lexical unit that consists of letters;
- <MIN> : matches any lower-case lexical unit;
- <MAJ> : matches any upper-case lexical unit;
- <PRE> : matches any lexical unit that consists of letters and starts with a capital letter.
- <DIC> : matches any word that is present in the dictionaries of the text;
- <SDIC> : matches any simple word in the text dictionaries;
- <CDIC> : matchesr any composed word in the dictionaries of the text;
- <NB> : matches any contiguous sequence of digit (1234 is matched but not 1 234);
- # : prohibits the presence of space.

4.3.2 References to the dictionaries

The second kind of patterns refer to the information in the dictionaries of the text.
The four possible forms are:

- `<lire>`: matches all the entries that have `lire` as canonical form;
- `<lire.V>`: matches all entries having `lire` as canonical form and have the grammatical code `V`;
- `<V>`: matches all entries having the grammatical code `V`;

- `{lirons,lire.V} ou <lirons,lire.V>`: matches all the entries having `lirons` as inflected form, `lire` as canonical form and the grammatical code `V`. That kind of pattern is only of interest if applied to the text automaton where all the ambiguities of the words are explicit.

  While executing a search on the text that pattern matches the same as the simple lexical unit `lirons`.

4.3.3 Grammatical and semantic constraints

The reference to the dictionary (`V`) in these examples is elementary. It's possible to express more complex patterns by indicating several grammatical or semantic codes separated by the character `+`. An entry of the dictionary is then only found if it has all the codes that are present in the pattern. The pattern `<N+z1>` thus recognizes the entries:

```
broderies,broderie.N+z1:fp
capitales européennes,capitale européenne.N+NA+Conc+HumColl+z1:fp
```

but not:

```
Descartes,René Descartes.N+Hum+NPropre:ms
habitué,.A+z1:ms
```

It is possible to exclude codes by preceding them with the character `-` instead of `+`. In order to be recognized an entry has to contain all the codes authorized by the pattern and none of the prohibited codes. The pattern `<A-z3>` thus recognizes all the adjectives that do not have the code `z3` (cf. table 3.2). If you want to refer to a code containing the character `-` you have to escape this character by preceding it with a `\`. Thus, the pattern `<N+faux\-ami>` could recognize all entries of the dictionaries containing the codes `N` and `faux-ami`.

The order in which the codes appear in the pattern is not important. The three following patterns are equivalent:

```
<N-Hum+z1>
<z1+N-Hum>
<-Hum+z1+N>
```

NOTE: it is not possible to use a pattern that only has prohibited codes. `<-N>` and `<-A-z1>` are thus incorrect patterns.

4.3.4 Inflectional constraints

It is also possible to specify constraints about the inflectional codes. These constraints have to be preceded by at least one grammatical or semantic code. They are represented as inflectional codes present in the dictionaries.

These are some examples of patterns that use inflectional constraints:

- <A:m> recognizes a masculine adjective;
- <A:mp:f> recognizes a masculine plural or a feminin adjective;
- <V:2:2> recognizes a verb in the 2nd or 3rd person; that excludes all tenses that have neither a 2nd or 3rd person (infinitive, past participle and present participle) as well as the tenses that are conjugated in the first person.

In order to let a dictionary entry E be recognized by pattern M, it is necessary that at least one inflectional code of E contains all the characters of an inflectional code of M. Consider the following example:

E=sépare,séparer.V+z1:P1s:P3s:S1s:S3s:Y2s
M=<V:P2s:Y2>

No inflectional code of E contains the characters P, 2 and s at the same time. However, the code Y2s of E does contain the characters Y and 2. The code Y2 is included in at least on code of E, the pattern M thus recognizes the entry E. The order of the characters inside an inflectional code is without importance.

4.3.5 Negation of a pattern

It is possible to negate a pattern by placing the character ! immediately after the character <.

Negation is possible with the patterns <MOT>, <MIN>, <MAJ>, <PRE>, <DIC> as well as with the patterns that carry grammatical, semantic of inflectional codes (*i.e.* <!V-z3:P3>). The patterns # and " " are each the negation of the other. The pattern <!MOT> can recognize all lexical units that do not consist of letters except for the phrase separator.

The negation is interpreted in a special way in the patterns <!DIC>, <!MIN>, <!MAJ> and <!PRE>. Instead of recognizing all forms that are not recognized by the pattern without negation these patterns only find forms that are sequences of letters. Thus, the pattern <!DIC> allows to find all unknown words in a text. These unknown forms are mostly proper name, neologisms and spelling errors.

These are several examples off patterns that mix the different types of constraints:

- <A-Hum:fs> : a non-human adjective in feminine singular;
- <lire.V:P:F> : the verb *lire* in present tense or future;
- <suis,suivre.V> : the word *suis* as inflected form of the verb *suivre* (as opposed the form of the verb *être*;
- <facteur.N-Hum> : all nominal entries that have *facteur* as canonical form and that do not have the semantic code Hum;

Concordance: file:///E:\My Unitex\French\Corpus\La peau de chagrin_snt\concord.html
200 matches
main, monsieur. - A l'instant, répondit Nathan. - Allons! allons! vous êtes deux braves. - Vous
droit, peut-être! reprit le belliqueux Nathan en se dressant comme un cerf-volant indécis. Il
er une joyeuse vie à la Panurge ou more orientali, couchés sur de moelleux coussins. Nous te de
pagne virginal de quelque jeune fille d'Otaïti, sa brûlante imagination lui peignait la vie sim
les étaient dues au génie de Bernard de Palissy; puis il dit à l'étranger d'un air insouciant:
ndit les escaliers en sifflant di tanti palpiti d'un souffle si faible, qu'il en entendit à pei
vine bouteille et sert de conclusion au Pantagruel. - Nous devons au Pater noster, répondit Rap
atin, et de passer une joyeuse vie à la Panurge ou more orientali, couchés sur de moelleux cous
ns vous donner un centime de France, un parat du Levant, un tarain de Sicile, un heller d'Allem
flamboyant aux yeux de saint Jean dans Pathmos. Une multitude de figures endolories, gracieuse
e ans pour une truite, pour un conte de Perrault ou une croquade de Charlet. - Vous avez bien r
lexandre sur un camée, les massacres de Pizarre dans une arquebuse à mèche, les guerres de reli
-Live. Le jeune homme contempla Senatus Populusque romanus: le consul, les licteurs, les toges
rgile d'un vase étrusque devant le Dieu Priape qu'elle saluait d'un air joyeux? En regard, une
environ. A ton respectable hôtel Saint-Quentin, dont par parenthèse l'enseigne inamovible offr
marchant. De force ou de bonne volonté, Raphaël fut entouré de ses amis, qui, l'ayant enchaîné

Figure 4.1: Result of the search for <!DIC>

- <!ADV> : all words that are not adverbes;
- <!MOT> : all symbols that are not letters except for the phrase separator (cf. figure 4.2).

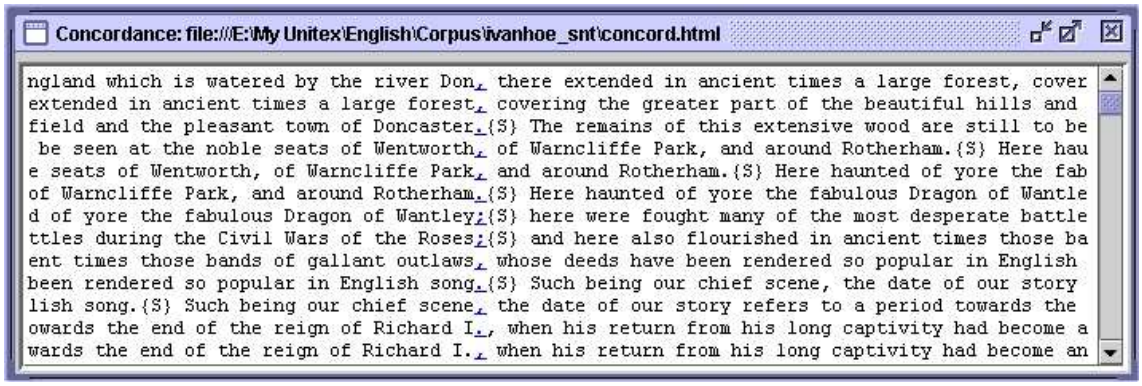
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
ngland which is watered by the river Don, there extended in ancient times a large forest, cover
extended in ancient times a large forest, covering the greater part of the beautiful hills and
field and the pleasant town of Doncaster.{S} The remains of this extensive wood are still to be
be seen at the noble seats of Wentworth, of Warncliffe Park, and around Rotherham.{S} Here hau
e seats of Wentworth, of Warncliffe Park, and around Rotherham.{S} Here haunted of yore the fab
of Warncliffe Park, and around Rotherham.{S} Here haunted of yore the fabulous Dragon of Wantle
d of yore the fabulous Dragon of Wantley;{S} here were fought many of the most desperate battle
ttles during the Civil Wars of the Roses;{S} and here also flourished in ancient times those ba
ent times those bands of gallant outlaws, whose deeds have been rendered so popular in English
been rendered so popular in English song.{S} Such being our chief scene, the date of our story
lish song.{S} Such being our chief scene, the date of our story refers to a period towards the
owards the end of the reign of Richard I., when his return from his long captivity had become a
wards the end of the reign of Richard I., when his return from his long captivity had become an

Figure 4.2: Result of a search for the pattern <!MOT>

4.4 Concatenation

There are three ways to concatenate regular expressions. The first consists in using the concatenation operator which is represented by the point. Thus, the expression:

<DET>.<N>

recognizes a determinator followed by a noun. The space can also be used for concatenation. The following expression:

```
le <A> chat
```

recognizes the lexical unit *le*, followed by an adjective an the lexical unit *chat*. Finally, it is possible to omit the point and the space using an opening bracket or the character < as well as after a closing bracket or after the character >. The brackets are used as delimiters of a regular expression. All of the following expression are equivalent:

```
le <A> chat
(le <A>)chat
le.<A>.chat
(le)<A>.chat
(le(<A>))(chat)
```

4.5 Union

The union of regular expressions is done by putting the character + between them. The expression:

```
(je+tu+il+elle+on+nous+vous+ils+elles) <V>
```

recognizes a pronoun followed by a verb. If an element of an expression should be optional it is sufficient to use the union of this element and the empty word epsilon.

Examples:

`le(petit+<E>)chat` recognizes the sequences *le chat* and *le petit chat*

`(<E>+franco-)(anglais+belge)` recognizes *anglais*, *belge*, *franco-anglais* and *franco-belge*

4.6 Kleene star

The Kleene star, represented by the character *, allows for recognizing zero, one or several occurrences of an expression. The star must be placed on the right hand side of the respective element. The expression:

```
il fait très* froid
```

recognizes *il fait froid*, *il fait très froid*, *il fait très très froid*, etc. The star has a higher priority than the other operators. You have to use brackets in order to apply the star to a complex expression. The expression:

```
0,(0+1+2+3+4+5+6+7+8+9)*
```

recognizes a zero followed by a comma and by a possibly empty sequence of digits.

ATTENTION: It is prohibited to search for the empty word with a regular expression. If you try to search for `(0+1+2+3+4+5+6+7+8+9)*`, the program will flag an error as shown in figure 4.3.

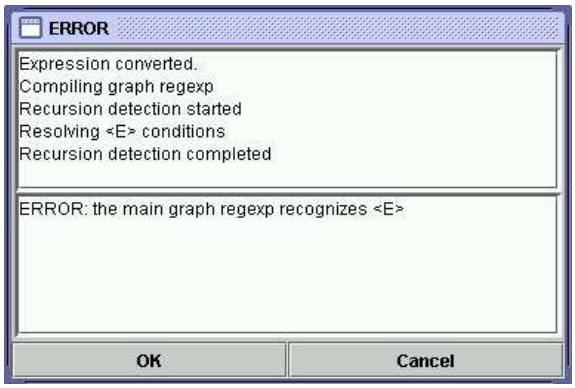

Figure 4.3: Error message when searching for the empty word

4.7 Search

4.7.1 Configuration of the search

In order to search for an expression you have to open a text at first (cf. chapter 2). Then click on "Locate Pattern..." in the menu "Text". The window of figure 4.4 appears.

The box "Locate pattern in the form of" allows to select regular expression or grammar. Click on "Regular expression".

The box "Index" allows to select the recognition mode:

- "Shortest matches" : prefer short matches;
- "Longest matches" : prefer longer matches. This is the default;
- "All matches" : Output all recognized sequences.

The box "Search limitation" to limit (or not) to a certain number of occurrences. By default, the search is limited to the 200 first occurrences.

The options of the box "Grammar outputs" do not concern the regular expressions. They are described in the section 6.4.

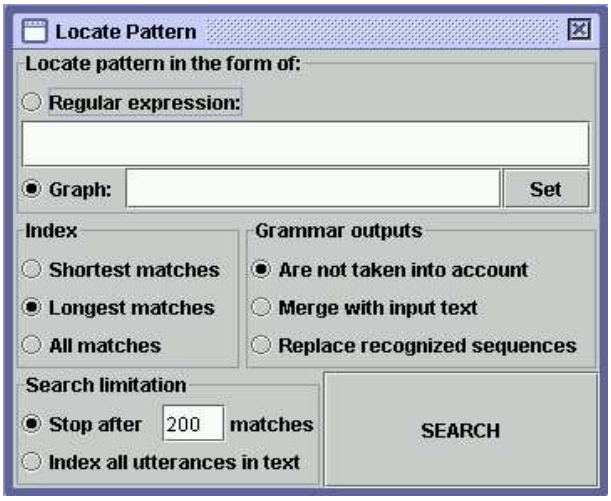

Figure 4.4: Window search for expressions

Enter an expression and click on "Search" in order to start the search. Unitex will transform the expression in a grammar in the format `.grf` . This grammar will then be compiled into a grammar of the format `.fst2` that will be used for the search.

4.7.2 Presentation of the results

When the search is finished the window of figure 4.5 appears showing the number of matched occurrences, the number of recognized lexical entities and the ratio between this number and the total number of lexical units in the text.

Figure 4.5: Search results

After having clicked on "OK" you will see window 4.6 appear, which allows to configure the presentation of the matched occurrences. You can also open this window by clicking on "Display Located Sequences..." in the menu "Text". We call the list of occurrences *concordance*.

The box "Modify text" offers the possibility to replace the matched occurrences with the generated outputs. This possibility will be examined in chapter 6.

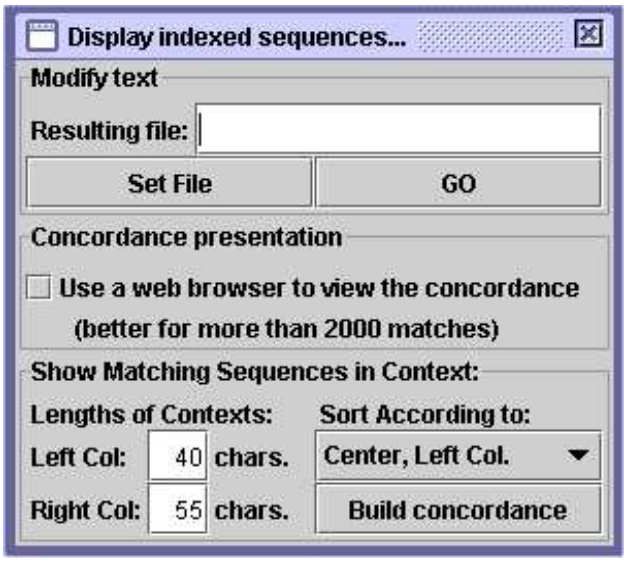

Figure 4.6: Configuration of the presentation of the found occurrences

In the box "Show Matching Sequences in Context" you can select the length in characters of the left and right contexts of the occurences that will be presented in the concordance. If an occurrence has less characters than its right context the line will be completed with the necessary number of characters. If an occurence has a length greater than that of the right context it will be displayed completely.

NOTE: in thaï, the size of the contexts is measured in displayable characters and not in real characters. This makes it possible to keep the line alinement in the concordance despite the presence of diacritics that combine with other letters instead of being displayed as normal characters.

You can choose the sort order in the list "Sort According to". The mode "Text Order" displays the occurrences in the order of their appearance in the text. The six other modes allow to sort in columns. The three zones of a line are the left context, the occurrence and the right context. The occurrences and the right contexts are sorted from left to right. The left contexts are sorted from right to left. The default mode is "Center, Left Col.". The concordance is generated in form of an HTML file.

If the concordances reaches several thousands of occurrances, it is advisable to display them in a web browser (Internet Explorer, Mozilla, Netscape, etc.). Check the box "Use a web browser o view the concordance" (cf. figure 4.6) in order to achieve that. This option is activated by default if the number of occurrences is above 3000. You can configure which web browser to use by clicking on "Preferences..." in the menu "Info". Click on the tab "Text Presentation" and select the program to use in the field "Html Viewer" (cf. figure 4.7).

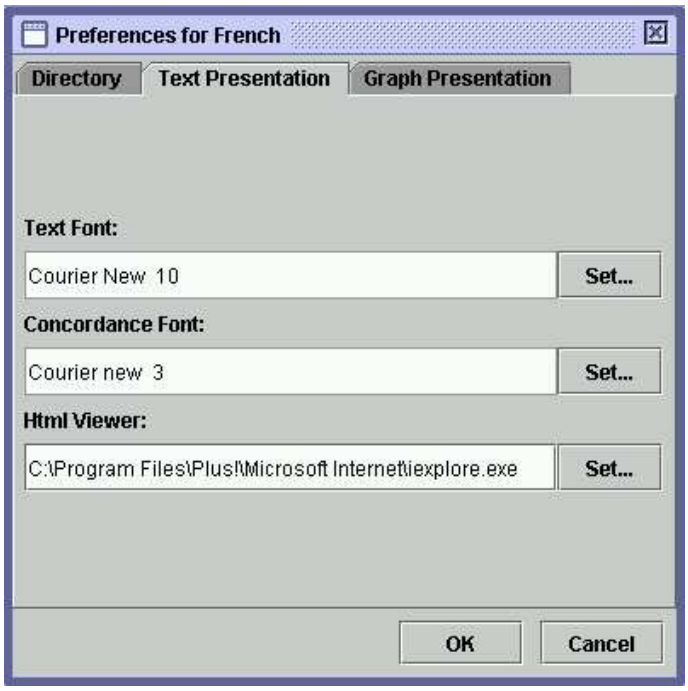

Figure 4.7: Selection of a web browser for displaying concordances

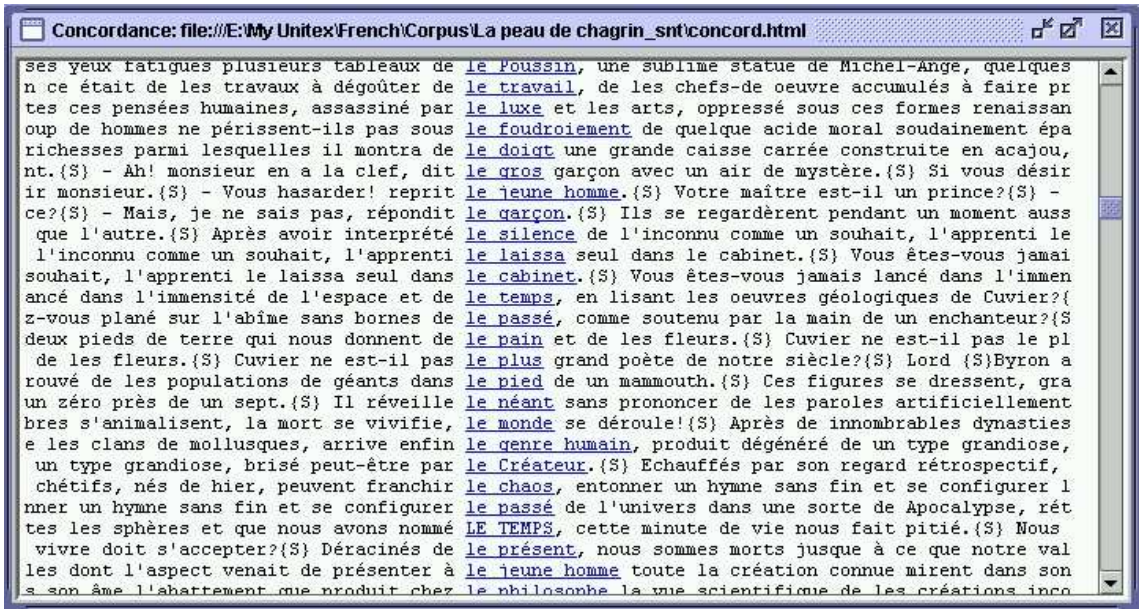
Concordance: file:///E:\My Unitex\French\Corpus\La peau de chagrin_snt\concord.html
ses yeux fatigués plusieurs tableaux de le Poussin, une sublime statue de Michel-Ange, quelques
n ce était de les travaux à dégoûter de le travail, de les chefs-de oeuvre accumulés à faire pr
tes ces pensées humaines, assassiné par le luxe et les arts, oppressé sous ces formes renaissan
oup de hommes ne périssent-ils pas sous le foudroiement de quelque acide moral soudainement épa
richesses parmi lesquelles il montra de le doigt une grande caisse carrée construite en acajou,
nt.{S} - Ah! monsieur en a la clef, dit le gros garçon avec un air de mystère.{S} Si vous désir
ir monsieur.{S} - Vous hasarder! reprit le jeune homme.{S} Votre maître est-il un prince?{S} -
ce?{S} - Mais, je ne sais pas, répondit le garçon.{S} Ils se regardèrent pendant un moment auss
que l'autre.{S} Après avoir interprété le silence de l'inconnu comme un souhait, l'apprenti le
l'inconnu comme un souhait, l'apprenti le laissa seul dans le cabinet.{S} Vous êtes-vous jamai
souhait, l'apprenti le laissa seul dans le cabinet.{S} Vous êtes-vous jamais lancé dans l'immen
ancé dans l'immensité de l'espace et de le temps, en lisant les oeuvres géologiques de Cuvier?{
z-vous plané sur l'abîme sans bornes de le passé, comme soutenu par la main de un enchanteur?{S
deux pieds de terre qui nous donnent de le pain et de les fleurs.{S} Cuvier ne est-il pas le pl
de les fleurs.{S} Cuvier ne est-il pas le plus grand poète de notre siècle?{S} Lord {S}Byron a
rouvé de les populations de géants dans le pied de un mammouth.{S} Ces figures se dressent, gra
un zéro près de un sept.{S} Il réveille le néant sans prononcer de les paroles artificiellement
bres s'animalisent, la mort se vivifie, le monde se déroule!{S} Après de innombrables dynasties
e les clans de mollusques, arrive enfin le genre humain, produit dégénéré de un type grandiose,
un type grandiose, brisé peut-être par le Créateur.{S} Echauffés par son regard rétrospectif,
chétifs, nés de hier, peuvent franchir le chaos, entonner un hymne sans fin et se configurer l
nner un hymne sans fin et se configurer le passé de l'univers dans une sorte de Apocalypse, rét
tes les sphères et que nous avons nommé LE TEMPS, cette minute de vie nous fait pitié.{S} Nous
vivre doit s'accepter?{S} Déracinés de le présent, nous sommes morts jusque à ce que notre val
les dont l'aspect venait de présenter à le jeune homme toute la création connue mirent dans son
s son âme l'abattement que produit chez le philosophe la vue scientifique de les créations inco

Figure 4.8: Example concordance

Chapter 5

Local Grammars

Local grammars are a powerful tool to represent the majority of linguistic phenomenons. The first section presents the formalism in which these grammars are represented. Then we will see how to construct and present grammars using Unitex.

5.1 The Local Grammar Formalism

5.1.1 Algebraic Grammars

Unitex grammars are variants of algebraic grammars, also known as context-free grammars . An algebraic grammar consts of rewriting rules. Below you see a grammar that matches any number of a characters:

$S \rightarrow aS$
$S \rightarrow \varepsilon$

The symbols to the left of the rules are called *non-terminal symbols* since they can be replaced. Symbols that cannot be replaced by other rules are called *terminal symbols*. The items at the right side are sequences of non-terminal and terminal symbols. The epsilon symbol ε designates the empty word. In the grammar above, S is a non-terminal symbol and a a terminal. S can be rewritten as either an a followed by a S or as the empty word. The operation of rewriting by applying a rule is called *derivation*. We say that a grammar recognizes a word if there exist a sequence of derivations that produce that word. The non-terminal that is the starting point of the first derivation is called an *axiom*.

The grammar above also recognizes the word *aa*, since we can get this word according to the axiom S by applying the following derivations:

Derivation 1: rewriting the axiom to aS
$\underline{S} \rightarrow aS$

Derivation 2: rewriting S at the right side of aS
$S \rightarrow a\underline{S} \rightarrow aaS$

Derivation 3: rewriting S to ε
$S \rightarrow aS \rightarrow aa\underline{S} \rightarrow aa$

We call the set of words recognized by a grammar the *grammar language*. The languages recognized by algebraic grammars are called *algebraic langiages*

5.1.2 Extended Algebraic Grammars

The extended algebraic grammars are algebraic grammars where the members on the right side of the rule are not just sequences of symbols but rational expressions.. Thus, the grammar that recognizes a sequence of an arbitrary number of a can be written as a grammar consisting of one rule:

$S \rightarrow a^*$

These grammars, also called *recursive transition networks* (*RTN*) or *syntax diagrams*, are suited for a user-friendly graphical representation. Indeed, the right member of a rule can be represented as a graph whose name is the left member of the rule.

However, Unitex grammars are not exactly extended algebraic grammars, since they contain the notion of *transduction*. This notion, which can be expressed only badly in finite state automata, signifies that a grammar may produce some output. In a sorrow for clarity, we will use the terms grammar or graph. When a grammar produces outputs, we will use the term *transducer*, as an extension of the definition of a transducer in the domain of finite state automata.

5.2 Editing Graphs

5.2.1 Import of Intex Graphs

In order to be able to use Intex graphs in Unitex, they have to be converted to Unicode. The conversion procedure is the same as the ones for texts (see section 2.2). If you're using Microsoft Word to perform this conversion, make sure that the graph always has the `.grf` extension after the conversion, since it happens that the `txt+ extension is automatically appended` If a `.txt` extension was appended, remove it.

ATTENTION: A graph converted to Unicode that was used in Unitex cannot be used in Intex any longer.

In order to use it again in Intex, you have to convert the text to ASCII, for example using the `Uni2Asc` program.. In addition to this, you have to open the graph in a text editor and replace the first line:

```
#Unigraph
```

by the following line:

```
#FSGraph 4.0
```

5.2.2 Creating a Graph

In order to create a graph, click on "New" in the "FSGraph" menu. You will then see the window coming up as in figure 5.2. The symbol in arrow form is the *init state* of the graph.. The round symbol with a square is the *final state* of the graph. The grammar only recognizes expressions that are described along the paths between init and final state.

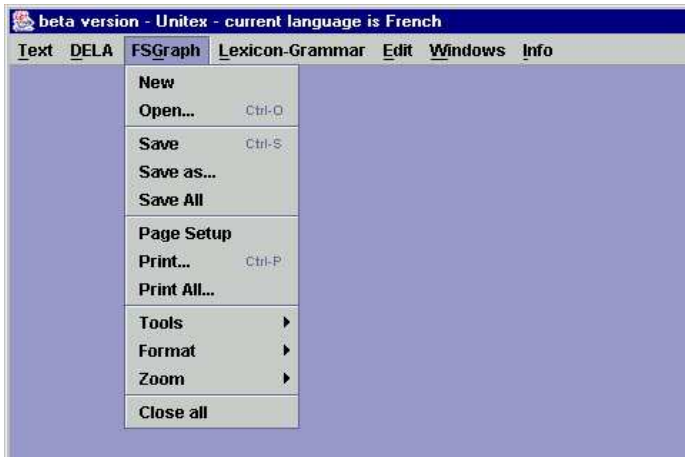

Figure 5.1: FSGraph Menu

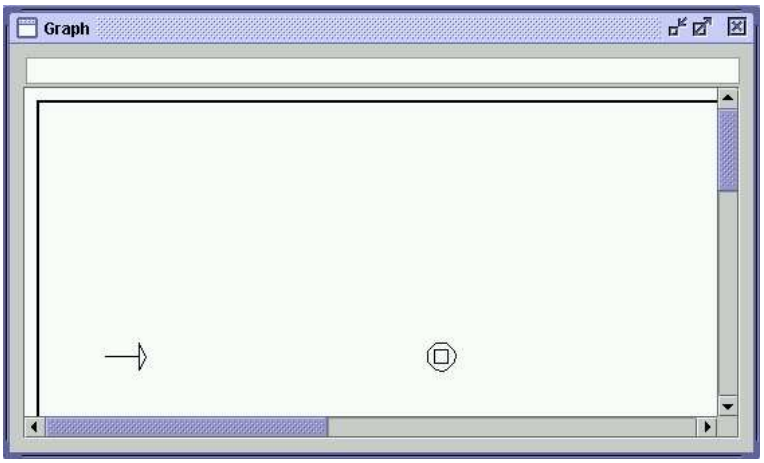

Figure 5.2: Blank Graph

In order to create a box, click inside of the window while pressing the Ctrl key. A blue rectangle will appear that symbolizes the empty box that was created (see figure 5.3). After creating the box, it is automatically selected.

You see the contents of that box in the text field at the top of the window. The newly created box contains the <E> symbol that represents the empty word epsilon.. Replace this

symbol by the text `le+la+l'+les` and press the enter key. You see that the box now contains four lines (see figure 5.4). The + character serves as a separator.. The box is displayed in the form of red text lines since it is not connected to another one at the moment. We often use this type of boxes to insert comments into a graph.

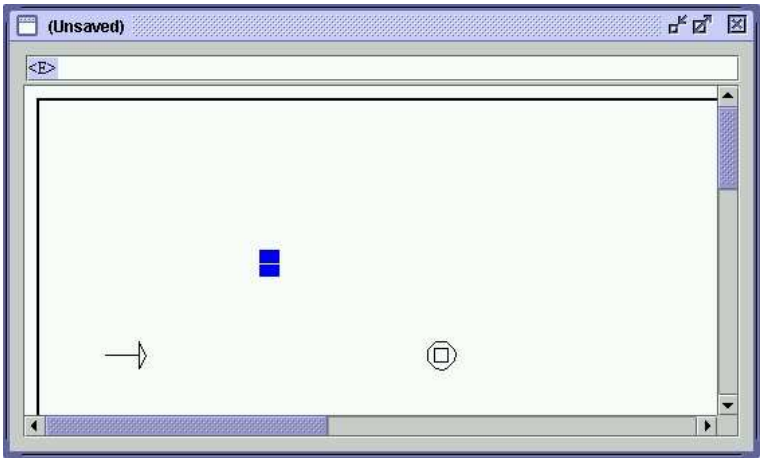

Figure 5.3: Creating a box

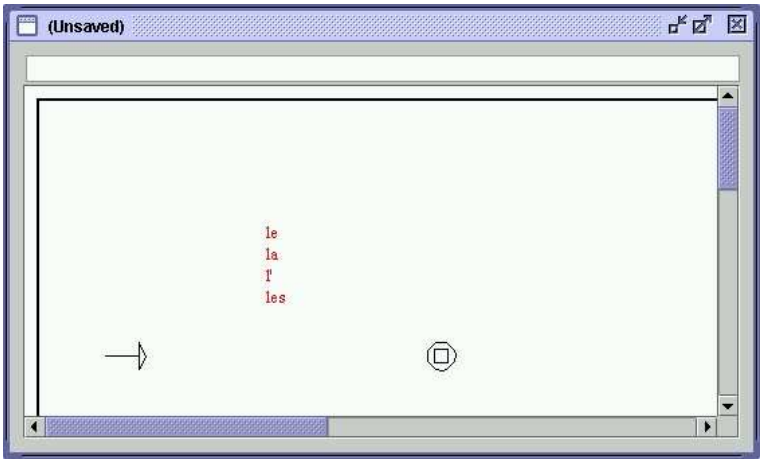

Figure 5.4: Box containing `le+la+l'+les`

To connect a box to another one, you have to first click on the source box, followed by a click on the target box.

If there already exists a transition between two boxes, it is deleted. It is also possible to use this operation by clicking first on the target box and then on the source box while pressing Shift. In our example, after connecting the box to the init and the final states of the graph, we get a graph like in figure 5.5:

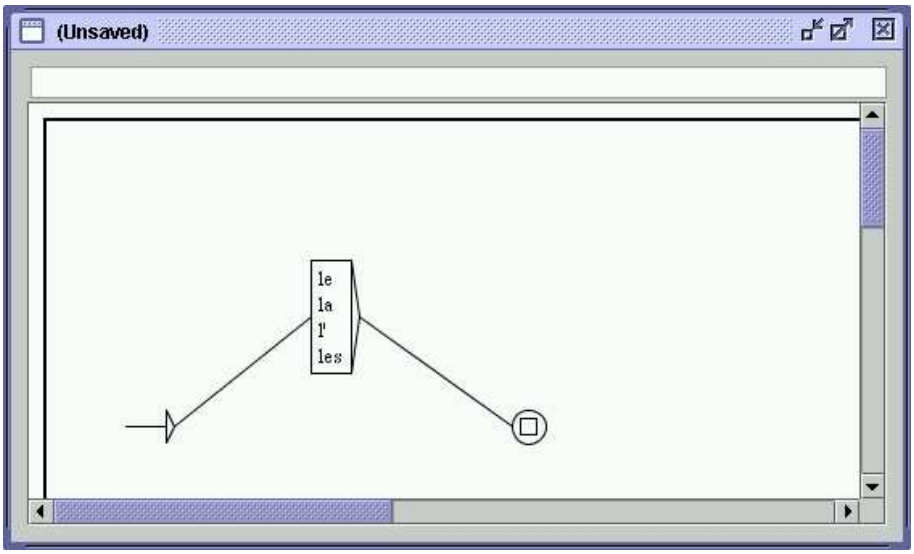

Figure 5.5: Graph that recognizes determiners in French

NOTE: If you double-click a box, you connect this box to itself (see figure 5.6). To undo this double-click on the same box a second time.

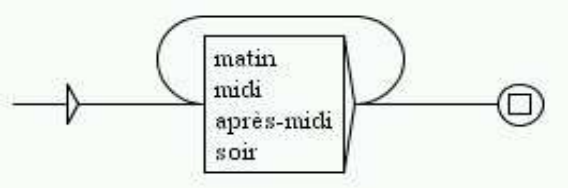

Figure 5.6: Box connected to itself

Click on "Save as..." in the "FSGraph" menu to save the graph.

By default, Unitex proposes to save the graph in the sub-directory `Graphs` in your personal folder. You can see if the graph was modified after the last saving if the title contains the text `(Unsaved)`.

5.2.3 Sub-Graphs

In order to call a sub-graph, its name is inserted into a box and preceded by the `:` character. If you enter the text `alpha+:beta+gamma+:e:\grec\delta.grf` into a box, you get a box similar to the one in figure 5.7:

You can indicate the complete name of the graph (`e:\grec\delta.grf`) or simply the name in the access path (`beta`); in this case, the the sub-graph is expected to be in the same directory as the graph that references it.

Figure 5.7: Graph that calls sub-graphs `beta` and `delta`

Calls to these sub-graphs are represented in the boxes by gray lines . On Windows you can open a sub-graph by clicking on the gray line while pressing the Alt key. On Linux, the combination <Alt+Click> is intercepted by the system. In order to open a sub-graph, click on its name by pressing the left and the right mouse button simultaneously.

5.2.4 Manipulating boxes

You can select several boxes using the mouse. In order to do so, click and drag the mouse without releasing the button. When you release the button, all boxes touched by the selection rectangle will be selected and are displayed in white on blue ground:

Figure 5.8: Selecting multiple boxes

When the boxes are selected, you can move them by clicking and dragging the cursor without releasing the button. In order to cancel the selection, click on an empty area of the graph. If you click on a box, all boxes of the selection will be connected to it.

You can perform a copy-paste using several boxes. Select them and press <Ctrl+C> or click on "Copy" in the "Edit" menu. The selection is now in the Unitex clipboard.

You can then paste this selection by pressing <Ctrl+V> or by selecting "Paste" in the "Edit" menu.

NOTE: You can paste a multiple selection into a different graph than the one where you copied it from.

In order to delete boxes, select them and delete the text that they contain. Delete the text presented in the text field above the window and press the enter key. The init and final states cannot be deleted.

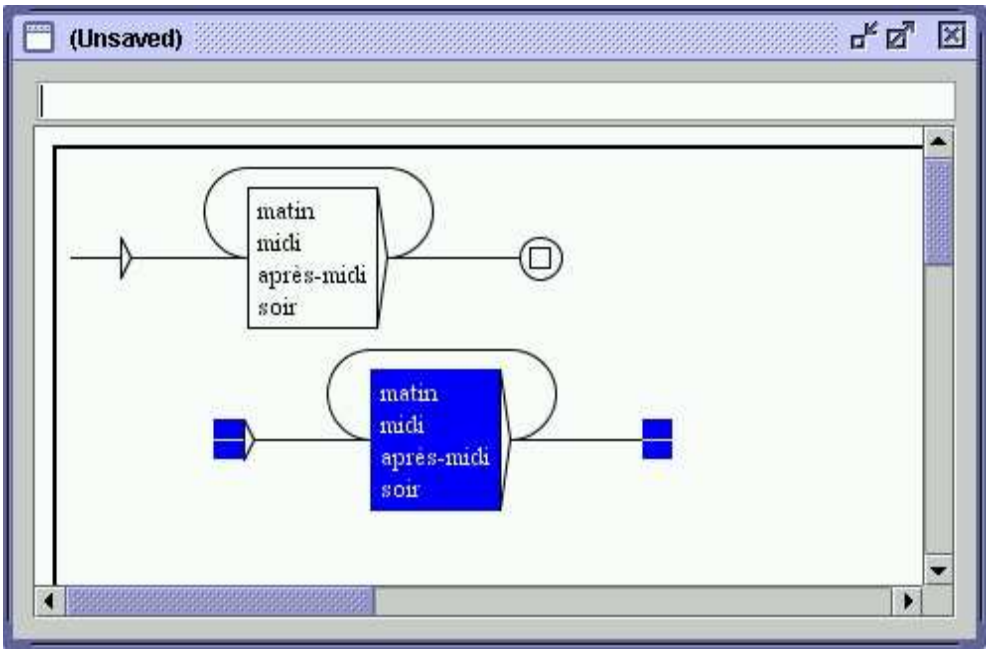

Figure 5.9: Copy-Paste of a multiple selection

5.2.5 Transducers

A transduction is an output associated with a box. To insert a transduction, use the / special character. All characters to the right of it will be part of the transduction. This, the text `un+deux+trois/nombre` results in a box like in figure 5.10:

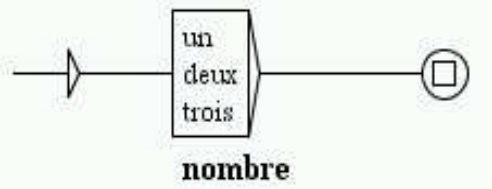

Figure 5.10: Example of a transduction

The transduction associated with a box is represented in bold text below it.

5.2.6 Using Variables

It is possible to select parts of of a recognized text by a grammar using varibales. To associate a variable `var1` with parts of a grammar, use the special symbols `$var1(` and `$var1)` to define the beginning and the end of the part to store. Create two boxes containing one `$var1(` and the second `$var1)`. These boxes must not contain anything but the variable

name preceded by `$` and followed by a parenthesis. Then link these boxes to the zone of the grammar to store. The graph in figure 5.11, you see a sequence beginning with an uppercase letter after `Monsieur` or `M.` that will be saved in a variable named `var1`.

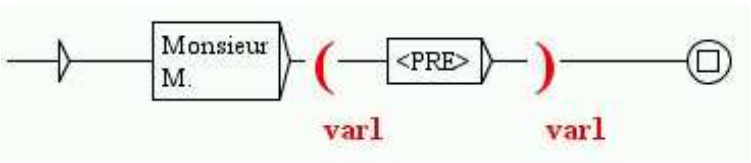

Figure 5.11: Using a variable `var1`

The variable names may contain letters (without accents), upper- or lowercase, numbers, or the _ (underscore) character. Unitex distinguishes between uppercase and lowercase characters.

When a variable is defined, you can use it in transductions by preceding its name with $. The grammar in figure 5.12 recognizes a date formed by a month and a year, and produces the same date as an output, but in the order year-month.

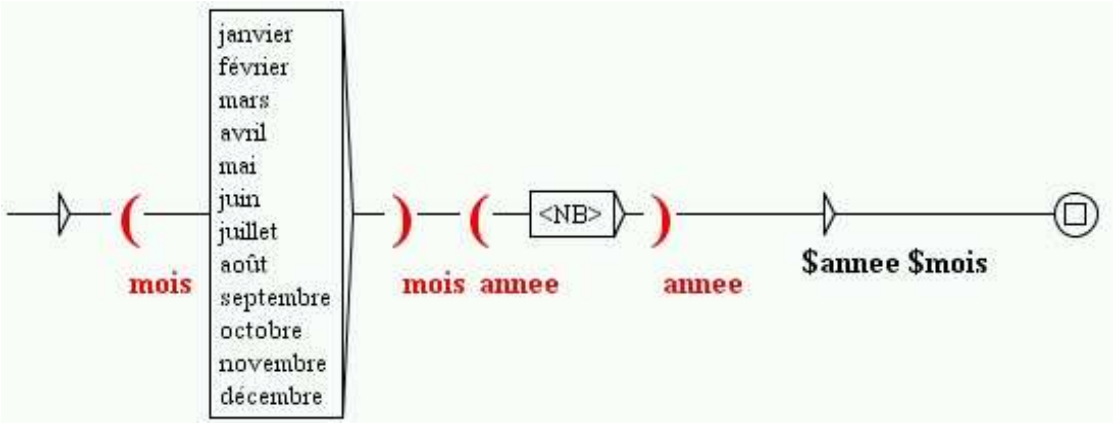

Figure 5.12: Inverting month and year in a date

5.2.7 Copying Lists

It can be practical to perform a copy-paste operation on a list of words or expressions from a text editor to a box in a graph. In order to avoid having to copy every term manually, Unitex provides a means to copy lists. To use this, select the list in your text editor and copy it using <Ctrl+C> or the copy function integrated in your editor. Then create a box in your graph, and press <Ctrl+V> or use the "Paste" command in the "Edit" menu to paste it into the box. A window as in figure 5.13 opens:

Figure 5.13: Selecting a context for copying a list

This window allows you to define the left and right contexts that will automatically be used for each term of the list. By default, these contexts are empty. If you use the contexts `<` and `.V>` with the following list:

manger
dormir
boire
jouer
lire

you wil get the box in figure 5.14:

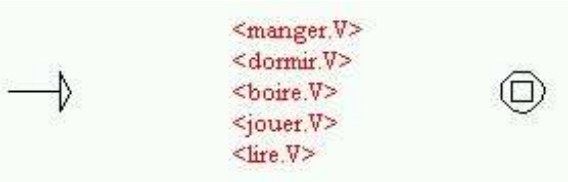

Figure 5.14: Box resulted from copying a list and applying contexts

5.2.8 Special Symbols

The Unitex graph editor interprets the following symbol in a special manner:

```
" + : / < > # \
```

Table 5.1 summarizes the meaning of these symbols for Unitex, as well as the places where these characters are recognized in the texts

Caracter	Meaning	Escape
"	quotation marks mark sequences that must not be interpreted by Unitex, and whose case must be taken verbatim	\"
+	+ separates different lines within the boxes	"+"
:	: introduces a call to a subgraph	":" ou \:
/	/ indicates the start of a transduction within a box	\/
<	< indicates the start of a pattern or a ??méta??	"<" ou \<
>	> indicates the end of a pattern or a ??méta??	">" ou \>
#	# forbids the presence of a space	"#"
\	\ escapes most of the special characters	\\

Table 5.1: encoding of special characters in the graph editor

5.2.9 Toolbar Commands

The toolbar to the left of the graphs contains short cuts for certain commands and allows for manipulating boxes of a graph by using some "utilities". This toolbar may be moved by clicking on the "rough" zone. It may also be dissociated from the graph and appear in an separate window (see figure 5.15). In this case, closing this window puts the toolbar pack at its initial position. Each graph contains its own toolbar.

Figure 5.15: Toolbar

The first two icons are shortcuts for saving and compiling the graph. The three following correspond to the "Copy", "Cut" and "Paste" operations. The last icon showing a ??key?? is a shortcut for the window for the graph display options.

The 6 icons left correspond to edit commands for boxes. The first one, forming a white arrow, corresponds to the boxes' normal edit mode. The 5 others correspond to some utilities. In order to use a utility, click on the corresponding icon: The mouse cursor changes its form and mouse clicks are then interpreted in a particular fashion. What follows is a description of these utiltities, from left to right:

- creating boxes: creates a box at the empty place where the mouse was clicked;
- deleting boxes: deletes the box that you click on;

- connect boxes to another box: using this utility you select one or more boxes and connect it or them to another one. In contrast to the normal mode, the connections are inserted to the box where the mouse button was released on;

- connect boxes to another box in the opposite direction: this utility performs the same operation as the one described above, but connects the boxes to the one clicked on in opposite direction;

- open a sub-graph: opens a sub-graph when you click on a grey line within a box.

5.3 Display options

5.3.1 Sorting the lines of a box

You can sort the contents of a box by selecting it and clicking on "Sort Node Label" in the "Tools" submenu of the "FSGraph" menu. This sort operation doesn't use the `SortTxt` program. It uses a basic sort mechanism that sorts the lines of the box according to the order of the characters in the Unicode encoding.

5.3.2 Zoom

The "Zoom" submenu allows you to choose the zoom scale that is applied to display the graph.

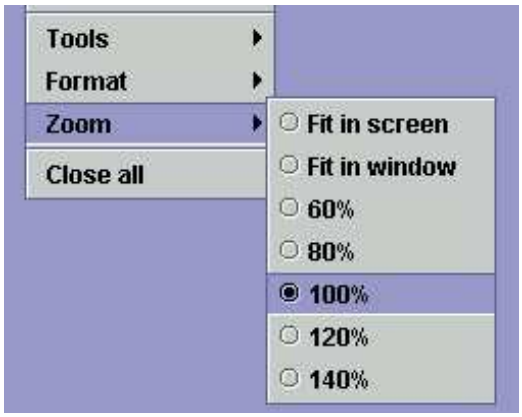

Figure 5.16: Zoom Sub-Menu

The option "Fit in screen" stretches or shrinks the graph in order to fit it into the screen. The option "Fit in window" adjusts the graph such that it is displayed entirely in the window.

5.3.3 Antialiasing

Antialiasing is a shading effect that avoids pixellisation effects. You can activate this effect by clicking on "Antialiasing..." in the "Format" sub-menu. Figure 5.17 shows one graph displayed normally (the graph on top) and with antialising (the graph at the bottom).

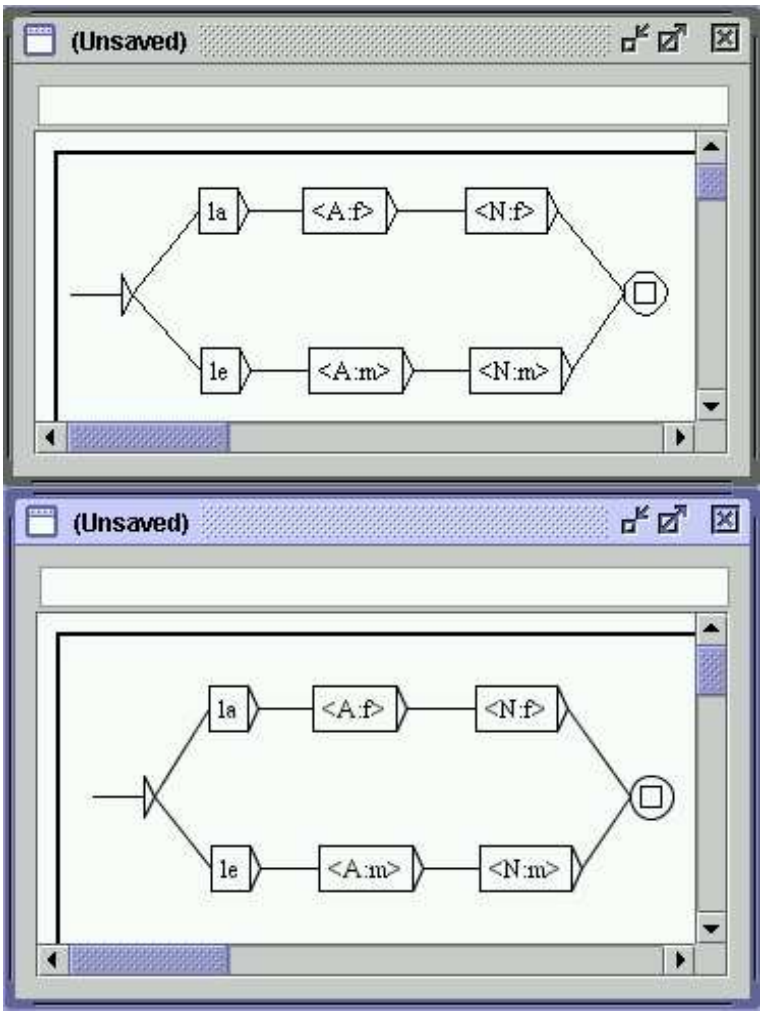

Figure 5.17: Antialiasing example

This effect slows Unitex down. We recommend you to not use it if your machine is not powerful enough.

5.3.4 Box alignment

In order to get harmoneous graphs, it is useful to align the boxes, either horizontally or vertically. To do this, select the boxes to align and click on "Alignment..." in the "Format"

sub menu of the "FSGraph" menu or press <Ctrl+M>. You will then see the window in figure 5.18.

The possibilities for horizontal alignment are:

- Top: The boxes are aligned to the top-most box;
- Center: The boxes are centered around the same axe;
- Bottom: The boxes are aligned to the bottom-most box.

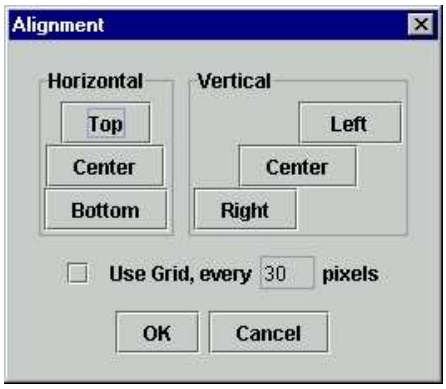

Figure 5.18: Alignment window

The possibilities for vertical alignment are:

- Left: The boxes are aligned to the left-most box;
- Center: The boxes are centered around the same axe;
- Right: The boxes are aligned to the right-most box.

Figure 5.19 shows an example for alignment. The group of boxes to the right is a copy of the ones to the left that was aligned vertically to the left.

The option "Use Grid" in the alignment window applies a grid to the background of the graph. This allows to approximately align the boxes.

5.3.5 Display, Options and Colors

You can configure the display style of a graph by pressing <Ctrl+R> or by clicking on "Presentation..." in the "Format" sub-menu of the "FSGraph" menu, which opens the window as in figure 5.21.

The font parameters are:

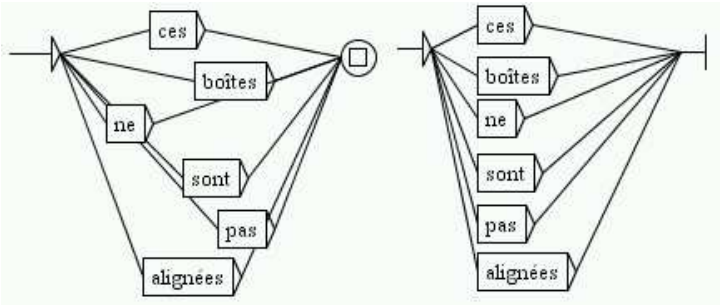

Figure 5.19: Example for aligning vertically to the left

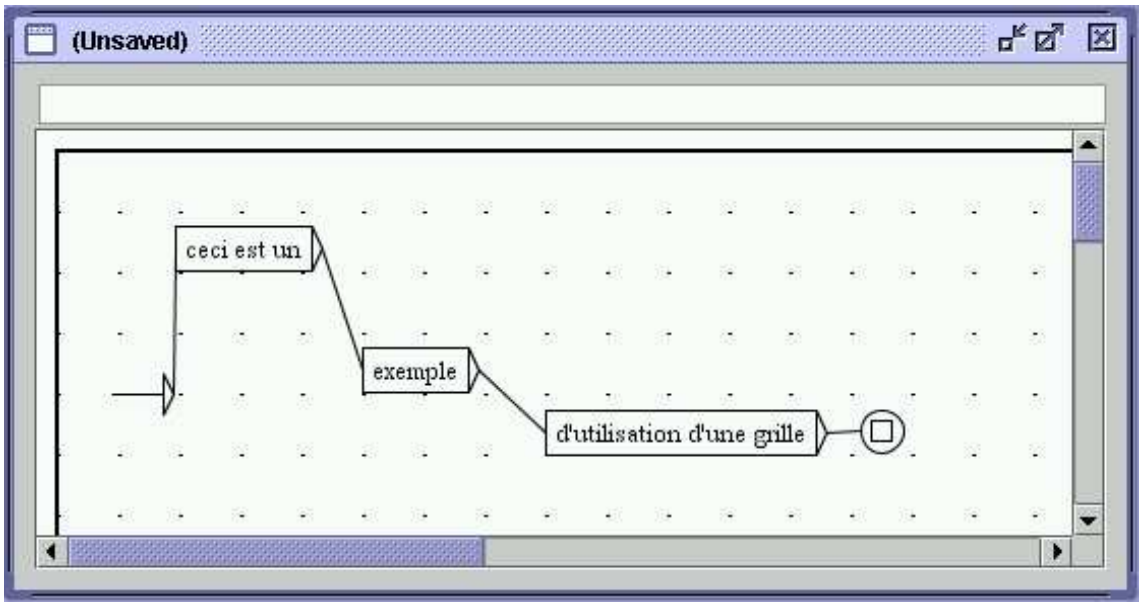

Figure 5.20: Example for using the grid

- Input: Font used within the boxes and in the text area where the contents of the boxes is edited;
- Output: font used for the attached transductions.

The color parameters are:

- Background: the background color;
- Foreground: the color used for the text and for the box display;
- Auxiliary Nodes: the color used for calls to sub-graphs;
- Selected Nodes: the color used for selected boxes;

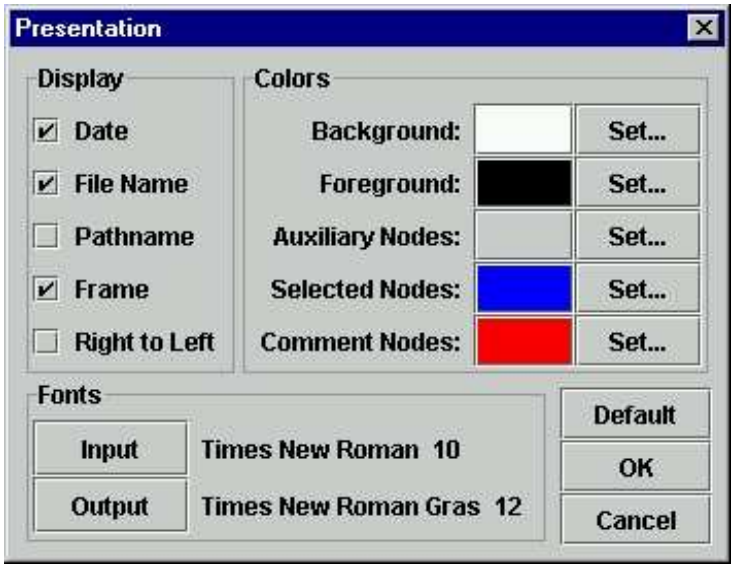

Figure 5.21: Configuring the display options of a graph

- Comment Nodes: the color used for boxes that are not connected to others.

The other parameters are:

- Date: display of the current date in the lower left corner of the graph;
- File Name: display of the graph name in the lower left corner of the graph;
- Pathame: display of the graph name along with its complete path in the lower left corner of the graph. This option doesn't have an effect if the option "File Name" is selected;
- Frame: draw a frame around the graph;
- Right to Left: invert the reading direction of the graph (see an example in figure 5.22).

You can reset the parameters to the default ones by clicking on "Default". If you click on "OK", only the current graph will be modified. . In order to modify the preferences for a language as a default, click on "Preferences..." in the "Info" menu and choose the tab "Graph Representaion".

The preferences configuration window has an extra option concerning antialiasing (see figure 5.23). This option activates antialiasing by default for all graphs in the current language. It is advised to not activate this option if your machine is not very fast.

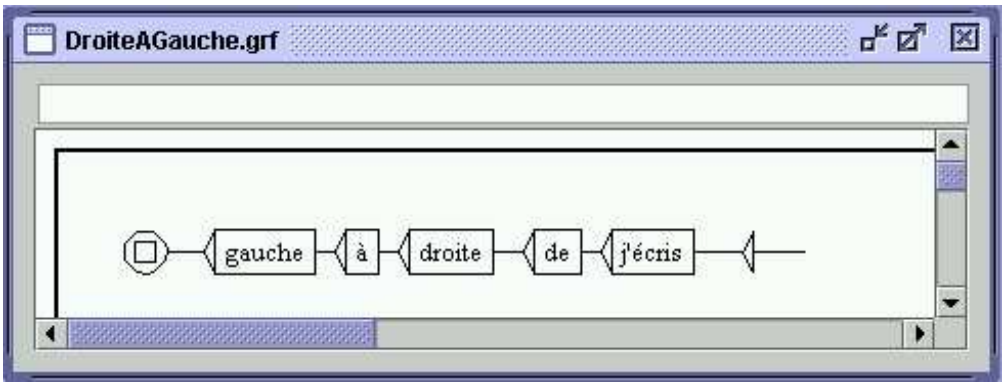

Figure 5.22: Graph with reading direction set to right to left

Figure 5.23: Default preferences configuration

5.4 Graphs outside of Unitex

5.4.1 Inserting a graph into a document

In order to include a graph into a document, you have to convert it into an image. To do this, activate antialiasing for the graph that interests you (this is not obligatory but results in

a better image quality).

On Windows:

Press "Print Screen" on your keyboard. This key should be next to the F12 key. Start the `Paint` program in the Windows "Utilities" menu. Press <Ctrl+V>. `Paint` will tell you that the image in the clipboard is too large and asks if you want to enlargen the image. Click on "Yes". You can now edit the screen image. Select the area that interests you. To do so, change to select mode by clicking on the dashed rectangle symbol in the upper left corner of the window. You can now select the area of the image using the mouse. When you have selected the zone, press <Ctrl+C>. Your selection is now in the clipboard, you can now just move to your document and press <Ctrl+V> to paste your image.

On Linux:

Take a screen capture (for example using the program `xv`). Edit your image at once using a graphic editor (for example `TheGimp`), and paste your image in your document in the same way as on Windows.

5.4.2 Printing a Graph

You can print a graph by clicking on "Print..." in the "FSGraph" menu or by pressing <Ctrl+P>.

ATTENTION: You should make sure that the page orientation parameter (portrait or landscape) corresponds well to the orientation of your graph.

You can specify the printing preferences by clicking on "Page Setup" in the "FSGraph" menu. You can also print all open graphs by clicking on "Print All...".

Chapter 6

Advanced use of graphs

6.1 Types of graphs

Unitex can work with four types of graphs that correspond to the following uses: automatic inflection of dictionaries, preprocessing of texts, normalization of text automatons and search for patterns. These different types of graphs are not interpreted in the same way by Unitex. Certain operations like the transduction are allowed for some types and forbidden for others. In addition, the special symbols are not the same depending on the type of the graph. This section presents each type of graph and shows their peculiarities.

6.1.1 Inflection graphs

An inflection graph describes the morphological variation that is associated with a word class by assigning inflectional codes to each variant. The paths of such a graph describe the modification that have to be applied to the canonical forms so that the transduction contain the inflectional information that will be produced.

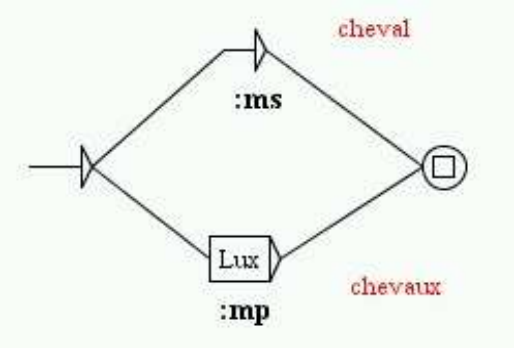

Figure 6.1: Example of an inflectional grammar

The paths may contain operators and letters. The possible operators are represented by the characters L and R. All letters that are not operators are characters. The only allowed spe-

cial symbol is the empty word <E>. It is not possible to refer to dictionaries in an inflection graph. It is also impossible to reference subgraphs.

Transductions are concatened in order to produce a string of characters. This string is then appended to the line of the produced dictionary (cf. chapter 3.4). The transductions with variables do not make sens in an inflection graph.

The contents of an inflection graph are manipulated without a change of case: the lowercase letters stay lowercase, the same for the uppercase letters. Besides, the connection of two boxes is exactly equivalent to the concatenation of their contents ??munie?? by the concatenation of their transductions. (cf. figure 6.2).

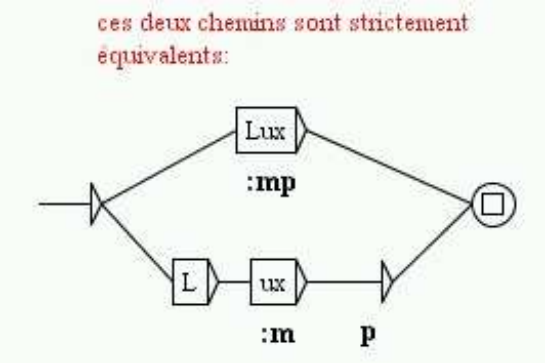

Figure 6.2: Two equivalent paths in an inflection grammar

The inflection graphs have to be compiled before being used by the inflection program.

6.1.2 Preprocessing graphs

Preprocessing graphs are meant to be applied to texts before they are tokenized into lexical units. These graphs can be used for inserting or replacing sequences in the texts. The two normal uses of these graphs are normalization of non-ambiguous forms and phrase boundary recognition.

The interpretation of these graphs in Unitex is very close to that of syntactic graphs used by the search for patterns. The differences are the following:

- you can use the special symbol <^> that recognizes a newline;
- it is impossible to refer to dictionaries;
- it is necessary to compile these graphs before they can be used for preprosessing operations.

The figures 2.6 and **??** show examples of preprocessing graphs.

6.1.3 Graphs for normalizing the text automaton

The graphs for normalization of the text automaton allow to normalize ambiguous forms. In fact they can describe several labels for the same form. These labels are then inserted into the text automaton thus making the ambiguities explicit. Figure 6.3 shows an extract of the normalization graph used for French.

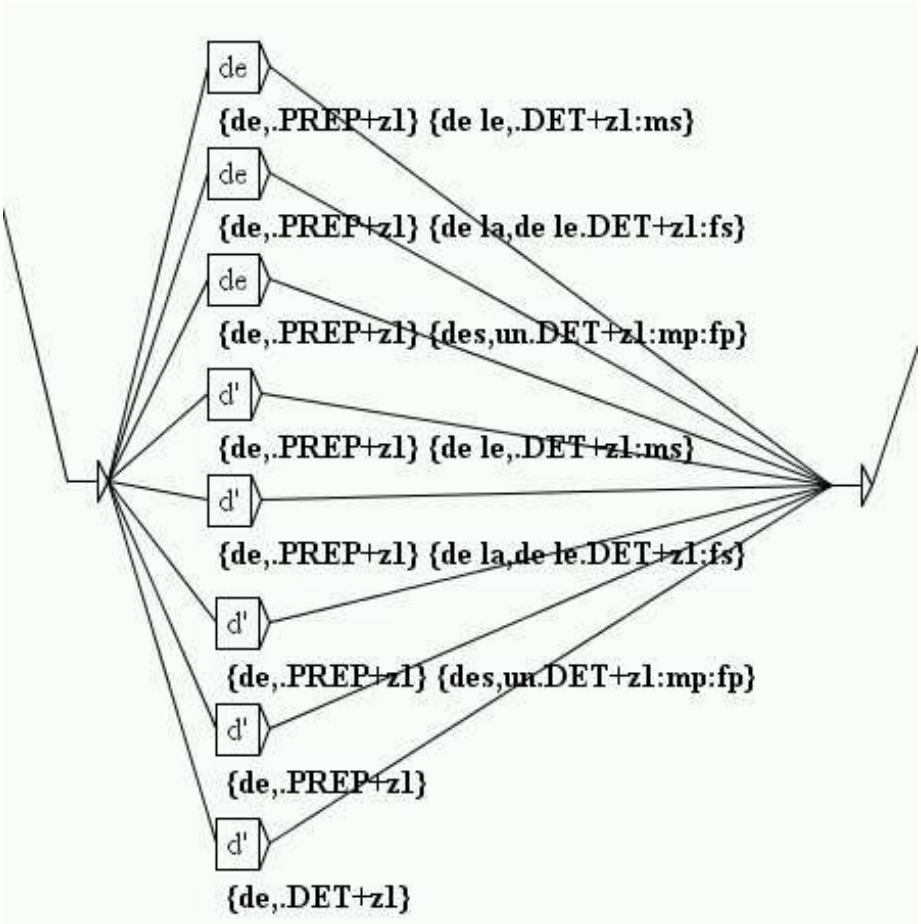

Figure 6.3: Extract of the normalisation graph used for French

The paths describe the forms that have to be normalized. The lowercase and uppercase variants are taken into account according to the following principle: the uppercase letters in the graph do not recognize the uppercase letters in the text automaton; the lowercase letters can recognize the lowercase and uppercase letters.

The transductions represent the sequence of label that will be inserted into the text automaton. These labels can be dictionary entries or strings of characters. The labels that represent entries of the dictionary have to respect the format for entries of a DELAF and are enclosed by the symbols { et }. The transductions with variables do not make sense in this kind of graph.

It is possible to reference subgraphs. It is not possible to reference dictionaries in order to describe the forms to normalize. The only special symbol that is recognized in this type of graph is the empty word `<E>`. The graphs for normalizing ambiguous forms should be compiled before using them.

6.1.4 Syntactic graphs

The syntactic graphs, often called local grammars, allow to describe syntactic patterns that could then be searched in the texts. Of all kinds of graphs these have the greatest expressional power because they allow to refer to dictionaries.

The lowercase/uppercase variants are authorized using the principle described above. It is still possible to enforce respect of case by enclosing an expression in quotes. The use of quotes also allows to enforce the respect of spaces. In fact, Unitex by default assumes that a space is possible between two boxes. In order to enforce the presence of a space you have to enclose it in quotes. For prohibiting the presence of a space you have to use the special symbol #.

The syntactic graphs can reference subgraphs (cf. section 5.2.3). They also have transductions including transductions with variables. The produced sequences are interpreted as strings of characters that will be inserted in the concordances or in the text if you want to modify it. (cf. section 6.4.3).

The special symbols that are supported by the syntactic graphs are the same that are usable in the regular expressions. (cf. section 4.3.1).

It is not obligatory to compile the syntactic graphs before using them by searching for patterns. If a graph is not compiled the system will compile it automatically.

6.1.5 Model graphs

The model graphs are meta-graphs that allow to generate a family of graphs starting from a lexical-grammatical table. It is possible to construct model graphs for all possible kinds of graphs. The construction and use of model graphs will be explained in chapter 8.

6.2 Compilation of a grammar

6.2.1 Compilation of a graph

The compilation is the operation that allows to pass from the format `.grf` to a format that can be manipulated easier by the Unitex programs. In order to compile a graph you have to open and then click on "Compile FST2" in the submenu "Tools" of the menu "FSGraph". Unitex then opens the program `Grf2Fst2`. You can keep track of its execution in a windows (cf. figure 6.4).

If the graph references subgraphs, those are automatically compiled. The result is a `.fst2` file that contains all the graphs that make up a grammar. The grammar is then ready to be used by the different Unitex programs.

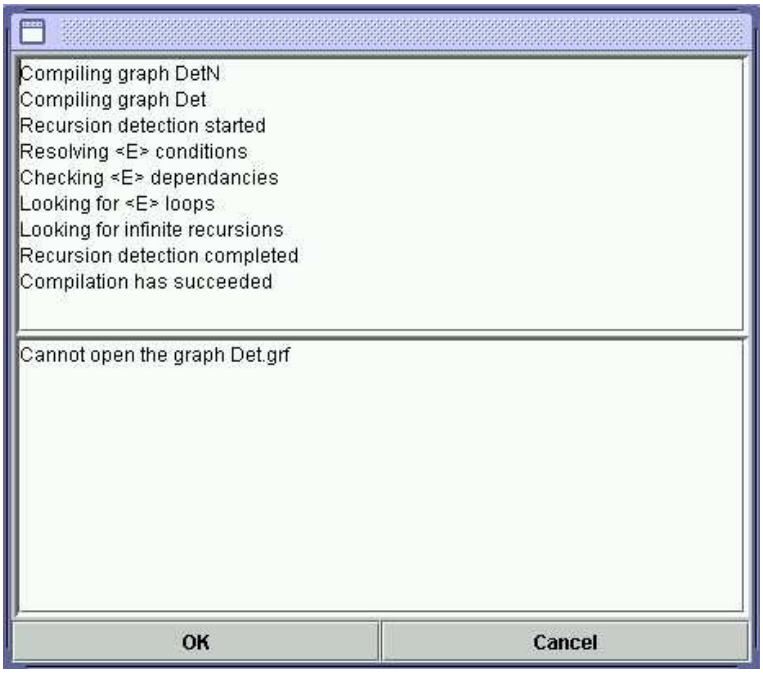

Figure 6.4: Compilation window

6.2.2 Approximation with a finite state transducer

The FST2 format conserves the architecture in subgraphs of the grammars, which is what makes them different from strict finite state transducers. The program `Flatten` allows to transform a grammar FST2 in a finite state transducer whenever this is possible and to construct and approximation if not. This function thus permits to obtain objects that are easier to manipulate and to which all classical algorithms on automatons can be applied.

In order to compile and thus transform a grammar select the command "Compile & Flatten FST2" in the submenu "Tools" of the menu "FSGraph". The windows of figure 6.5 allows you to configure the operation of approximation.

The box "Flattening depth" lets you specify the level of embedding of subgraphs. This value represents the maximum death above which the calling of subgraphs will no longer be replaced y the subgraphs themselves.

The box "Expected result grammar format" allows to determine the inclusion of the program above the indicate limit. If you select the option "Finite State Transducer", the calls to subgraphs will be ignored above the maximum depth. This option guarantees that a finite state transducer is obtained possibly not equivalent to the grammar of depart. On the contrary, the option "equivalent FST2" indicates that the program should allow for subgraph calls above the limited depth. This option guarantees strict equivalence of the result with the original grammar but does not forcibly reproduce a finite state transducer. This option can be used for optimizing certain grammars.

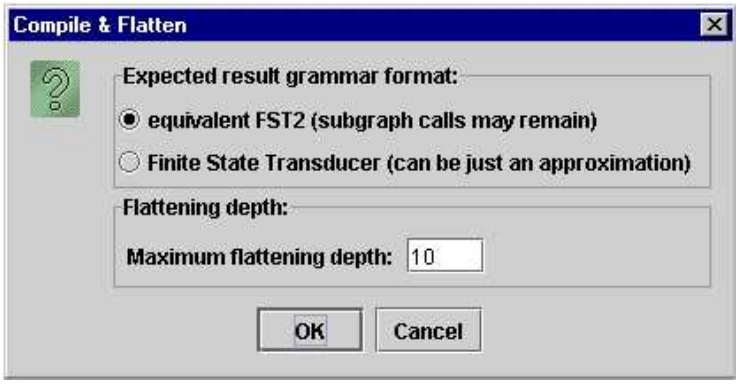

Figure 6.5: Configuration of approximation of a grammar

A message indicates at the end of the approximation process if the result is a finite state transducer or an FST2 grammar and in the case of a transducer if it is equivalent to the original grammar. (cf. figure 6.6).

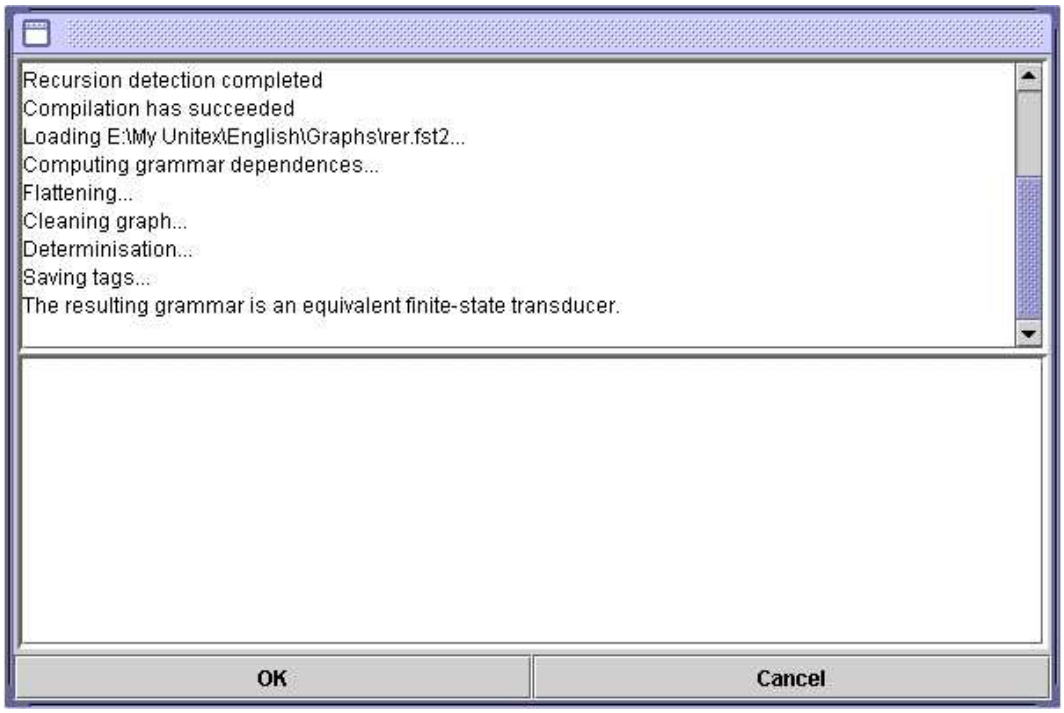

Figure 6.6: Resultat of the approximation of a grammar

6.2.3 Constraints on grammars

in saved inflection grammars a grammar can never have an empty path. This means that the principal path of a grammar may not recognize the empty word but this does not prevent a subgraph of that grammar to recognize epsilon.

It is not possible to associate a transduction to a call to a subgraph. Such transductions are ignored by Unitex. It is therefore necessary to use an empty box that is situated to left of the call to the subgraph in order to carry the transduction (cf. figure 6.7).

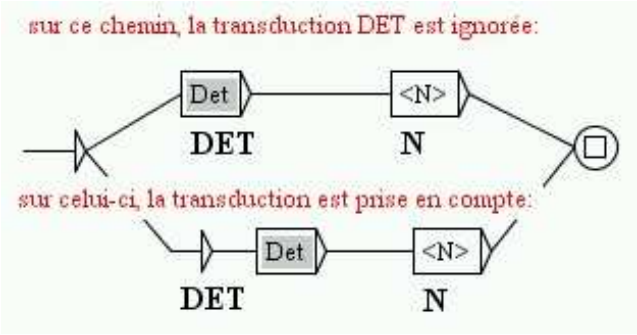

Figure 6.7: How to associate a transduction to a call to a subgraph

The grammars may not contain infinite loops because the Unitex programs cannot terminate the exploration of such a grammar. These loops can be due to transitions that are labeled by the empty word or by recursive calls to subgraphs.

The loop due to transitons with the empty words can have two origins of which the first is illustrated by the figure 6.8.

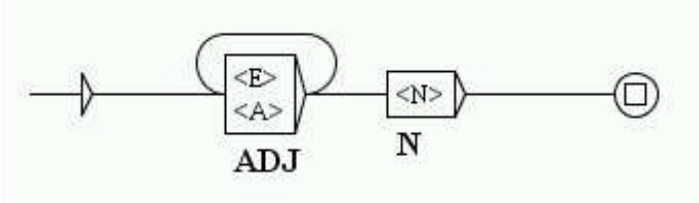

Figure 6.8: Infinite loops due to a transition by the empty word with a transductions

This type of loops is due to the fact that a transition with the empty word cannot be eliminated automatically by Unitex because it is associated with a transduction. Thus, the transition with the empty word of figure 6.8 will not be suppressed and will provoke an infinite loop.

The second category of loop by epsilon concerns the call to subgraphs that can recognize the empty word. This case is illustrated in figure 6.9: if the subgraph `Adj` recognizes epsilon, there is an infinite loop that Unitex cannot detect.

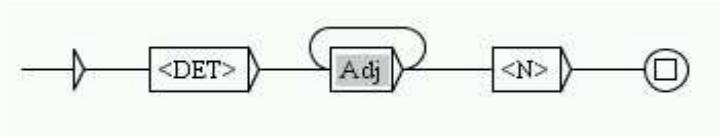

Figure 6.9: Infinite loop due to a call to a subgraph that recognizes epsilon

The third possibility of infinite loops lies in recursive calls to subgraphs. Look at the graphs `Det` and `DetCompose` in figure 6.10. Each of these graphs can call the other *without reading any text*. The fact that none of these two graphs has labels between the initial state and the call to the subgraph is ??capital??. In fact, if there was at least one label different from epsilon between the beginning of the graph `Det` and the call to `DetCompose`, this would mean that the Unitex programs that explore the graph `Det` would have to read the pattern described by that label in the text before calling `DetCompose` recursively. In this case the programs could not loop infinitely if they would not recognize the pattern an infinite number of times in the text.

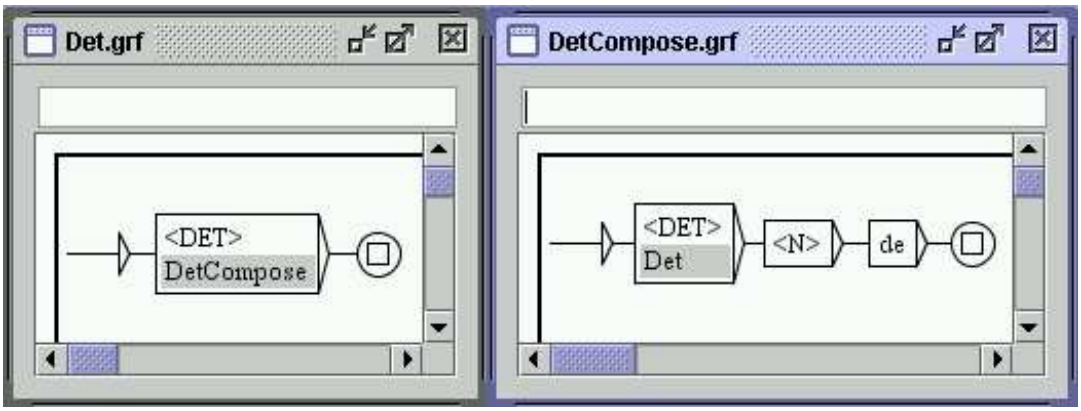

Figure 6.10: Infinite loop caused by two graphs calling each other

6.2.4 Error detection

In order to keep the programs from blocking or crashing, Unitex automatically detects errors during graph compilation. The graph compiler verifies that the principal graph does not recognize the empty word and searches for all possible forms of infinite loops. When an error is encountered an error message is displayed in the compilation window. Figure 6.11 shows the message that appears if one tries to compile the graph `Det` of figure 6.10.

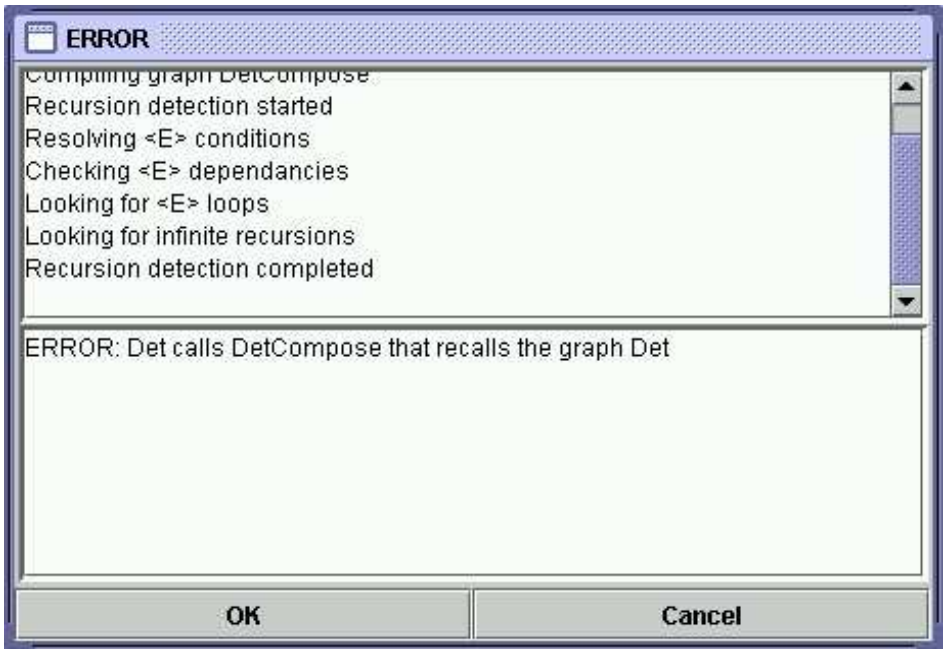

Figure 6.11: Error message when trying to compile `Det`

If you have started a pattern search be selecting a graph of the format `.grf` and Unitex discovers an error, the operation is automatically interrupted.

6.3 Rules for the application of transducers

This section describes the rules for the application of transducers along with the operaions of preprocessing and the search for patterns. The following does not apply to inflection graphs and normalization graphs for ambiguous forms.

6.3.1 Insertion to the left of the matched pattern

When a transducer is applied in REPLACE mode, the output replaces the sequences that have been read in the text. In MERGE mode, the output is inserted to the left of the recognized sequences. Look at the transducer in figure 6.12.

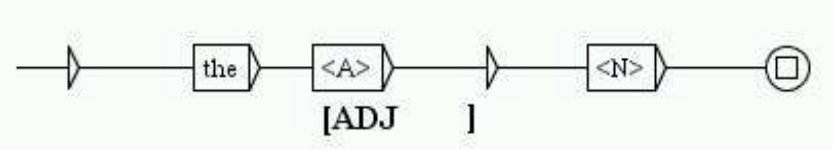

Figure 6.12: Example of a transducer

If this transducer is applied to the novel *Ivanhoe* by Sir Walter Scott in MERGE mode, the following concordance is obtained.

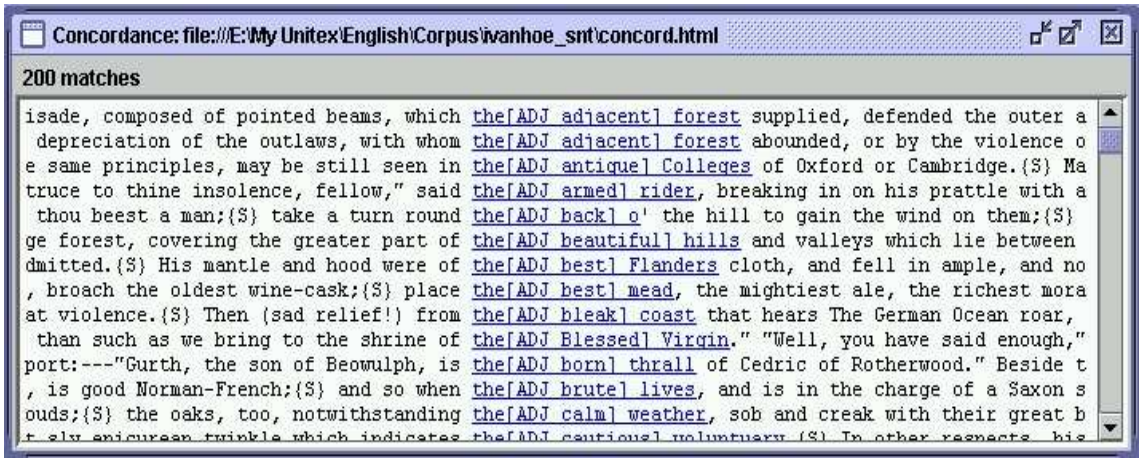
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
200 matches
isade, composed of pointed beams, which the[ADJ adjacent] forest supplied, defended the outer a
depreciation of the outlaws, with whom the[ADJ adjacent] forest abounded, or by the violence o
e same principles, may be still seen in the[ADJ antique] Colleges of Oxford or Cambridge.{S} Ma
truce to thine insolence, fellow," said the[ADJ armed] rider, breaking in on his prattle with a
thou beest a man;{S} take a turn round the[ADJ back] o' the hill to gain the wind on them;{S}
ge forest, covering the greater part of the[ADJ beautiful] hills and valleys which lie between
dmitted.{S} His mantle and hood were of the[ADJ best] Flanders cloth, and fell in ample, and no
, broach the oldest wine-cask;{S} place the[ADJ best] mead, the mightiest ale, the richest mora
at violence.{S} Then (sad relief!) from the[ADJ bleak] coast that hears The German Ocean roar,
than such as we bring to the shrine of the[ADJ Blessed] Virgin." "Well, you have said enough,"
port:---"Gurth, the son of Beowulph, is the[ADJ born] thrall of Cedric of Rotherwood." Beside t
, is good Norman-French;{S} and so when the[ADJ brute] lives, and is in the charge of a Saxon s
ouds;{S} the oaks, too, notwithstanding the[ADJ calm] weather, sob and creak with their great b

Figure 6.13: Concordance obtained in MERGE mode with the transducer of figure 6.12

6.3.2 Application while progressing

During the preprocessing operations, the text is modified while being read. In order to avoid the risk of infinite loops, it is necessary that the sequences that are produced by a transducer will not be re-analyzed by the same ones. Therefore, whenever a sequence is inserted into the text, the application of the transducer is continued after that sequence.

This rule does not apply the preprocessing transducers, because during the application of syntactic graphs, the transductions do not modify the processed text but a concordance file different from the text.

6.3.3 Priority of the leftmost match

During the application of a local grammar, the collected occurrences are all indexed. During the construction of the concordance all these occurrence are presented (cf. figure 6.14).

```
atered by the river Don, there extended [in ancient] times a large forest, covering
red by the river Don, there extended in [ancient times] a large forest, covering the
he river Don, there extended in ancient [times a] large forest, covering the greater
```

Figure 6.14: Occurrences are collected into concordance

In exchange, if you modify a text instead of constructing a concordance, it is necessary to choose among these occurrences those that will be taken into account. Unitex applies the following priorisation rule for that purpose: the leftmost sequence is used.

If this rule is applied to the three occurrrences of the preceeding concordance, the occurrence `[in ancient]` is concurrent `[ancient times]`. It is therefore the first that is

retained because this is the leftmost occurrence and `[ancient times]` is eliminated. The following occurrence of `[times a]` is no longer in conflict with `[ancient times]` and can therefore appear in the result:

```
...Don, there extended [in ancient] [times a] large forest...
```

The rule of priority of the leftmost match is applied only when the text is modified, be it during preprocessing, be it after the application of a syntactic graph (cf. section 6.4.3).

6.3.4 Priority of the longest match

During the application of a syntactic graph it is possible to choose if the priority should be given to the shortest or the longest sequences or if all sequences should be retained. During preprocessing, the priority is always given to the longest sequences.

6.3.5 Transductions with variables

As we have seen in section 5.2.6, it is possible to use variables to store the text that has been analyzed by a grammar. These variables can be used in the preprocessing graphs and in the syntactic graphs.

You have to give names to the variables you use. These names can contain non-accentuated lower-case and upper-case letters between A and Z, digits and the character _ (underscore).

In order to define the end of the zone that is stored in a variable, you have to create a box that contains the name of the variable enclosed in the characters $ and (($ and) for the end of a variable). In order to use a variable in a transduction, its name must be preceded by the character $ (cf. figure 6.15).

Variables are global. This means that you can to define a variable in a graph and and reference it in another as is illustrated in the graphs of figure 6.15:

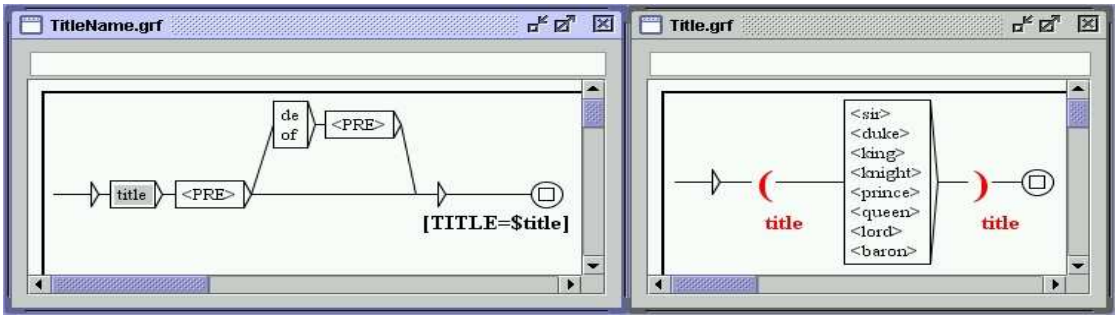

Figure 6.15: Definition of a variable in a subgraph

If the graph `TitleName` is applied in MERGE mode to the text *Ivanhoe*, the following concordance is obtained:

Transductions with variables can be used to move groups of words. In fact, the application of a transducer in REPLACE mode inserts only the produced sequences into the text.

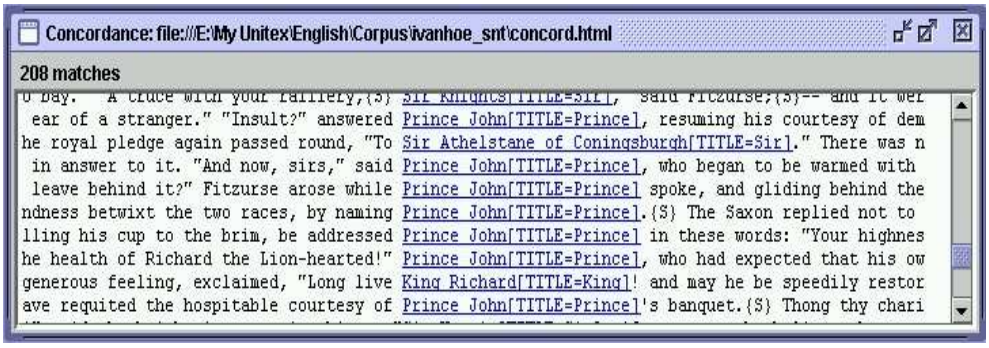
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
208 matches
ear of a stranger." "Insult?" answered Prince John[TITLE=Prince], resuming his courtesy of dem
he royal pledge again passed round, "To Sir Athelstane of Coningsburgh[TITLE=Sir]." There was n
in answer to it. "And now, sirs," said Prince John[TITLE=Prince], who began to be warmed with
leave behind it?" Fitzurse arose while Prince John[TITLE=Prince] spoke, and gliding behind the
ndness betwixt the two races, by naming Prince John[TITLE=Prince].{S} The Saxon replied not to
lling his cup to the brim, be addressed Prince John[TITLE=Prince] in these words: "Your highnes
he health of Richard the Lion-hearted!" Prince John[TITLE=Prince], who had expected that his ow
generous feeling, exclaimed, "Long live King Richard[TITLE=King]! and may he be speedily restor
ave requited the hospitable courtesy of Prince John[TITLE=Prince]'s banquet.{S} Thong thy chari

Figure 6.16: Concordance obtained by the application of the graph `TitleName`

In order to inverse two groups of words it is sufficient to store them into variables and produce a transduction with these variables in the desired order. Thus, the application of the transducer in figure 6.17 in REPLACE mode to the text *Ivanhoe* results in the concordance of figure 6.18.

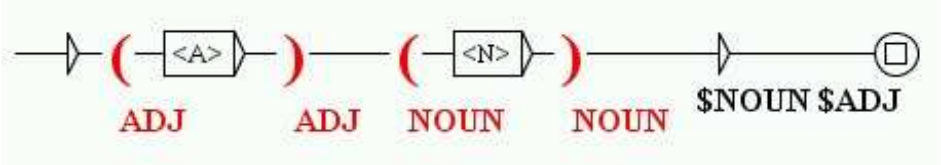

Figure 6.17: Inversion of words using two variables

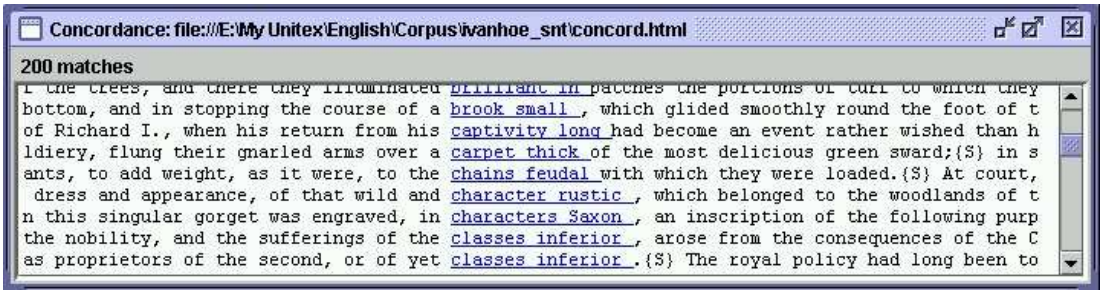
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
200 matches
bottom, and in stopping the course of a brook small , which glided smoothly round the foot of t
of Richard I., when his return from his captivity long had become an event rather wished than h
ldiery, flung their gnarled arms over a carpet thick of the most delicious green sward;{S} in s
ants, to add weight, as it were, to the chains feudal with which they were loaded.{S} At court,
dress and appearance, of that wild and character rustic , which belonged to the woodlands of t
n this singular gorget was engraved, in characters Saxon , an inscription of the following purp
the nobility, and the sufferings of the classes inferior , arose from the consequences of the C
as proprietors of the second, or of yet classes inferior .{S} The royal policy had long been to

Figure 6.18: Result of the application of the transducer in figure 6.17

The presence of a space to the right of each occurrence in the concordance of figure 6.18 is due to the insertion of a space after the `$NOUN $ADJ` in the transduction. Without this space, the result of the transduction would be collated to the right context (cf. figure 6.19).

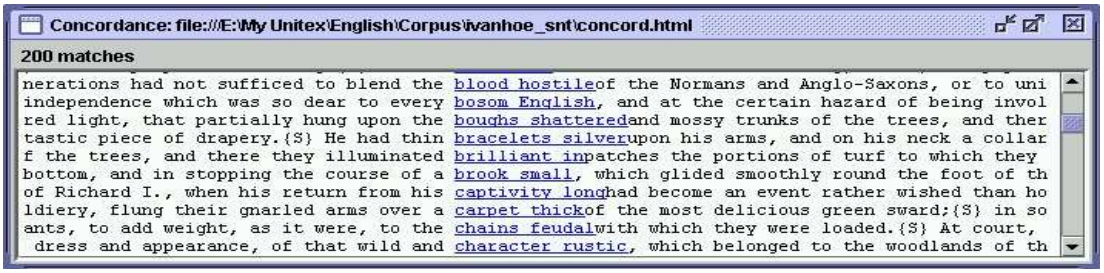
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
200 matches
nerations had not sufficed to blend the blood hostileof the Normans and Anglo-Saxons, or to uni
independence which was so dear to every bosom English, and at the certain hazard of being invol
red light, that partially hung upon the boughs shatteredand mossy trunks of the trees, and ther
tastic piece of drapery.{S} He had thin bracelets silverupon his arms, and on his neck a collar
f the trees, and there they illuminated brilliant inpatches the portions of turf to which they
bottom, and in stopping the course of a brook small, which glided smoothly round the foot of th
of Richard I., when his return from his captivity longhad become an event rather wished than ho
ldiery, flung their gnarled arms over a carpet thickof the most delicious green sward;{S} in so
ants, to add weight, as it were, to the chains feudalwith which they were loaded.{S} At court,
dress and appearance, of that wild and character rustic, which belonged to the woodlands of th

Figure 6.19: Spacing problem in REPLACE mode

In fact, the program `Locate` always considers the possibility of a facultative space between two boxes. In the present case the program tries to read a space between the box that constitutes the end of the variable `NOUN` and the box containing the transduction. If a space is read in REPLACE mode, it is erased because it is part of the text analyzed by the grammar. In order to avoid the loss of this space, it is therefore necessary to reinsert it by putting it into a transduction.

If the beginning or the end of variable is malformed (end of a variable before its beginning or absence of the beginning or end of a variable), it will be ignored during the transductions.

There is no limitation of the number of possible variables.

The variables can be overlapping even congruently as is shown in figure 6.20:

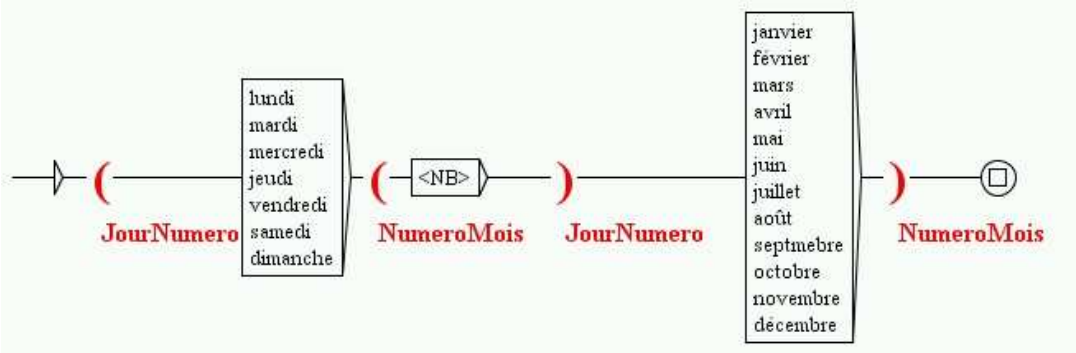

Figure 6.20: Congruent variables

6.4 Application of graphs to texts

This section only applies to syntactic graphs.

6.4.1 Configuration of the search

In order to apply a graph to a text, you have to open the text, then click on "Locate Pattern..." in the "Text" menu, or press <Ctrl+L>. You can then configure your search in the window shown in figure 6.21.

In the field "Locate pattern in the form of", choose "Graph" and select your graph by clicking on the "Set" button. You can choose a graph in `.grf` format (Unicode Graphs) or a compiled graph in `.fst2` format (Unicode Compiled Graphs). If your graph is in `.grf` format, Unitex will compile it automatically before starting the search.

The "Index" field allows to select the recognition mode.

- "Shortest matches" : give precedence to the shortest matches;
- "Longest matches" : give precedence to the longest sequences. This is the default mode;
- "All matches" : give out all recognized sequences.

The field "Search limitation" allows to limit the search to a certain number of occurrence or not. By default, the search is limited to the 200 first occurrences.

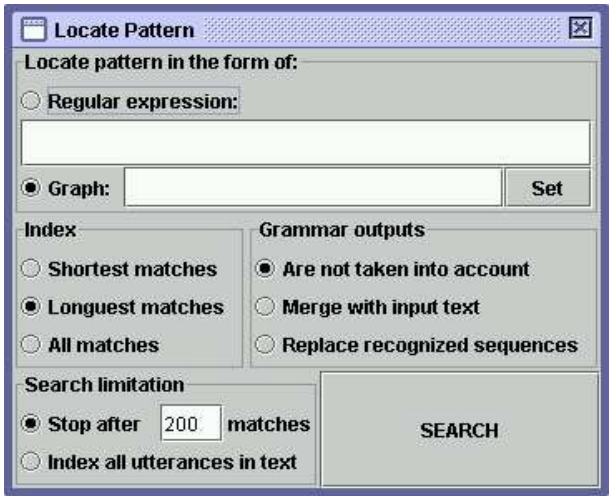

Figure 6.21: Expression search window

The field "Grammar outputs" concerns the use of the transductions. The mode "Merge with input text" allows to insert the sequences that are produced by the transductions. The mode "Replace recognized sequences" allows to replace the recognized sequences with the produced sequences. The third mode ignores all transductions. This latter mode is used by default.

After you have selected the parameters, click on "SEARCH" to start the search.

6.4.2 Concordance

The result of a search is an index file that contains the positions of all enountered occurrences. The window of figure 6.22 lets you choose wether to construct a concordance or modify the text.

In order to display a concordance, you have to click on the botton "Build concordance". You can parameterize the size of left and right contexts in characters. You can also choose the sorting mode that will be applied to the lines of the concordance in the menu "Sort According to". For further details on the parameters of concordance construction, refer to the section 4.7.2

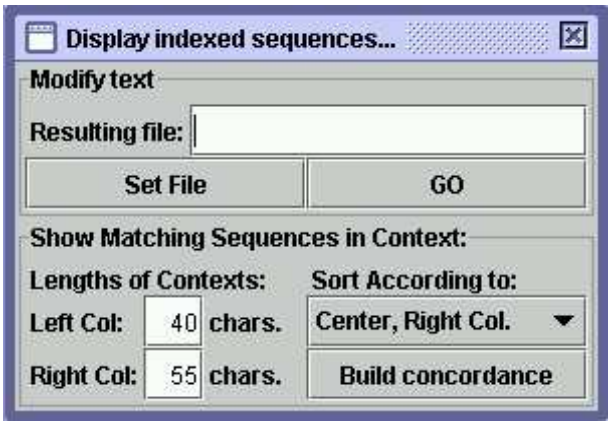

Figure 6.22: Configuration for displaying the encountered occurrences

The concordance is produced in form of an HTML file. You can parameterize Unitex so that the concordances will be read using a web browser (cf. section 4.7.2).

If you display the concordances with the window provided by Unitex you can access a recognized sequence in the text by clicking on the occurrence. If the text window is not iconified and the text is not too long to be displayed, you see the selected sequence appear (cf. figure 6.23).

Furthermore, if the text automaton has been constructed, and if the corresponding window is not iconified, clicking on an occurrence selects the automaton of the phrase that contains this occurrence.

6.4.3 Modification of the text

You can choose to modify the text instead of constructing a concordance. In order to do that choose a file name in the field "Modify text" in the window of figure 6.22. This file has to have the extension `.txt`.

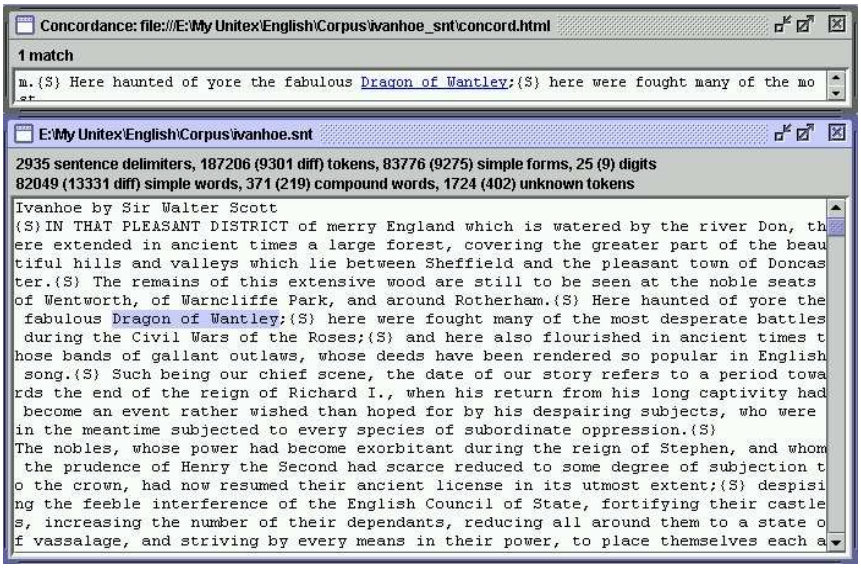
Concordance: file:///E:\My Unitex\English\Corpus\ivanhoe_snt\concord.html
1 match
m.{S} Here haunted of yore the fabulous Dragon of Wantley;{S} here were fought many of the mo
E:\My Unitex\English\Corpus\ivanhoe.snt
2935 sentence delimiters, 187206 (9301 diff) tokens, 83776 (9275) simple forms, 25 (9) digits
82049 (13331 diff) simple words, 371 (219) compound words, 1724 (402) unknown tokens
Ivanhoe by Sir Walter Scott
{S}IN THAT PLEASANT DISTRICT of merry England which is watered by the river Don, th
ere extended in ancient times a large forest, covering the greater part of the beau
tiful hills and valleys which lie between Sheffield and the pleasant town of Doncas
ter.{S} The remains of this extensive wood are still to be seen at the noble seats
of Wentworth, of Warncliffe Park, and around Rotherham.{S} Here haunted of yore the
fabulous Dragon of Wantley;{S} here were fought many of the most desperate battles
during the Civil Wars of the Roses;{S} and here also flourished in ancient times t
hose bands of gallant outlaws, whose deeds have been rendered so popular in English
song.{S} Such being our chief scene, the date of our story refers to a period towa
rds the end of the reign of Richard I., when his return from his long captivity had
become an event rather wished than hoped for by his despairing subjects, who were
in the meantime subjected to every species of subordinate oppression.{S}
The nobles, whose power had become exorbitant during the reign of Stephen, and whom
the prudence of Henry the Second had scarce reduced to some degree of subjection t
o the crown, had now resumed their ancient license in its utmost extent;{S} despisi
ng the feeble interference of the English Council of State, fortifying their castle
s, increasing the number of their dependants, reducing all around them to a state o
f vassalage, and striving by every means in their power, to place themselves each a

Figure 6.23: Selection of an occurrence in the text

If you want to modify the current text, you have to choose the corresponding `.txt` file. If you choose another file name, the current text will not be affected. Click on the "GO" button to start the modification of the text. The precedence rules that are applied during these operations are described in section 3.6.2.

After the operation has been started, the resulting file is a copy of the text in which all transductions have been taken into account. The normalization operations and the splitting into lexical units are automatically applied to this text file. The existing text dictionaries are not modified. Thus, if you have chosen to modify the current text, the modifications will be effective immediately. You can then start new searches on the text.

ATTENTION: if you have chosen to apply your graph ignoring the transductions, all occurrences will be erased from the text.

Chapter 7

Text automata

Natural languages contain lots of lexical ambiguities. The text automaton is an effective and visual means of representing these ambiguties. Each phrase of a text is represented by an automaton the paths of which express all possible interpretations.

This chapter presents the text automata, the details of their construction and the operations that can be applied. It is not possible at the moment to search for patterns on the text automaton nor to use rules in order to eliminate ambiguities.

7.1 Presentation

The text automaton can express all possible lexical interpretations of the words. These different interpretations are the different entires presented in the dictionary of the text. Figure 7.1 shows the automaton of the fourth phrase of the text *Ivanhoe*.

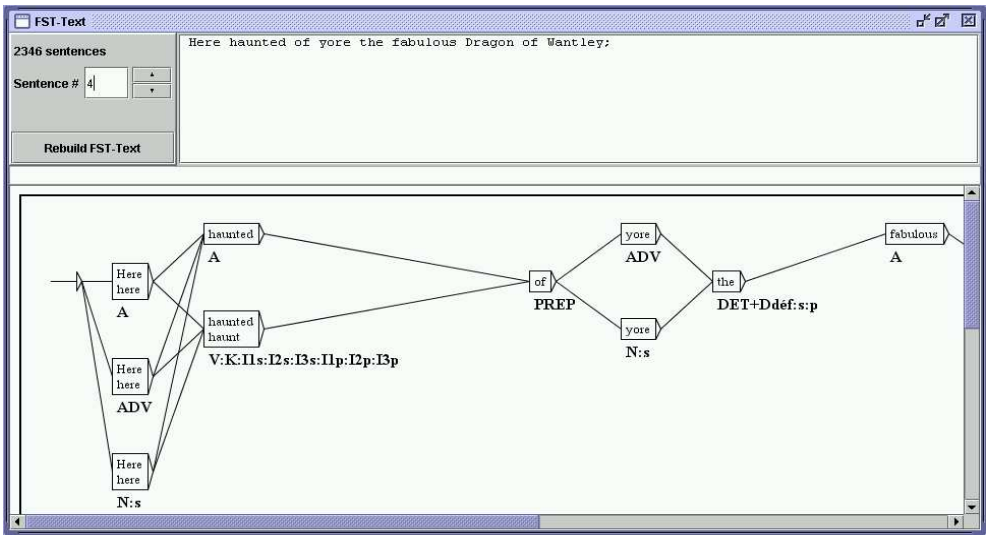

Figure 7.1: Example of the automaton of the phrase

You can see in figure 7.1 that the word `Here` possess three interpretations here (adjective, adverbe and noun), `haunted` two (adjective and verb), etc. All the possible combinations are expressed because each interpretation of each word is connected to all the interpretations of the following and preceding words.

In case of concurrence between a composite word and a sequence of simple words, the automaton contains a path that is labeled by the composite word, parallel to the paths that express the combinations of simple words. This is illustrated in figure 7.2, where the composite word `courts of law` is concurring with a combination of simple words.

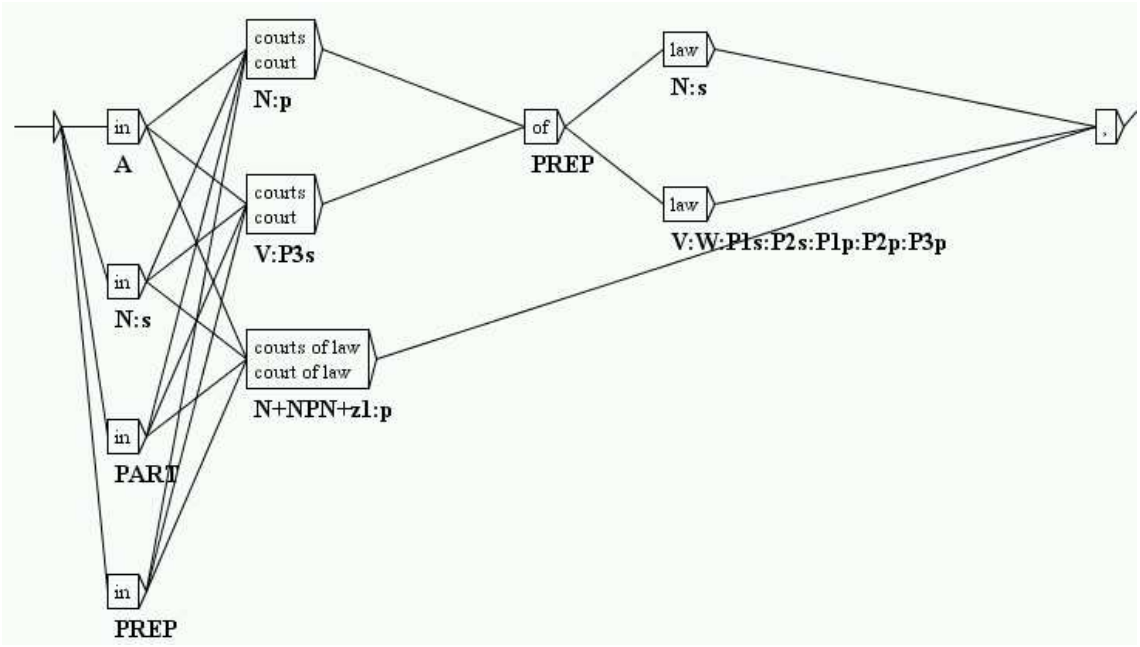

Figure 7.2: Concurrence between a composite word and a combination of simple words.

By construction, the automaton of the text doesn't contain any loops. One says that the text automaton is *acyclic*.

NOTE: the term text automaton is an abuse of the language. In fact, there is an automaton for each phrase of the text. Therefore, the combination of all these automata correspond to the automaton of the text. Therefore the term text automaton is used even if this object is not really manipulated for practical reasons.

7.2 Construction

In order to construct the text automaton you have to open this text, then click on "Construct FST-Text..." in the menu "Text". It is recommended to have split the text at sentence boundaries text and to have applied the dictionaries. If you have not applied sentence boundary detection, the construction program will split the text arbitrarily in sequences of 2000 lexical

units instead of construction one automaton per phrase. If have not applied the dictionaries, the phrase automaton that you obtain will consist of only one path made up only of unknown words.

7.2.1 Rules of construction of text automata

The phrase automata are constructed starting from the text dictionaries. The obtained degree of ambiguity is therefore directly linked to the granularity of the descriptions of the used dictionaries. From the phrase automaton in figure 7.3, you can conclude that the word `which` has been code twice as a determinator in two subcategories of the category `DET`. This granularity of descriptions will not be of any use if you are not interested in the grammatical category of this word. It is therefore necessary to adapt the granularity of the dictionaries to the intended use.

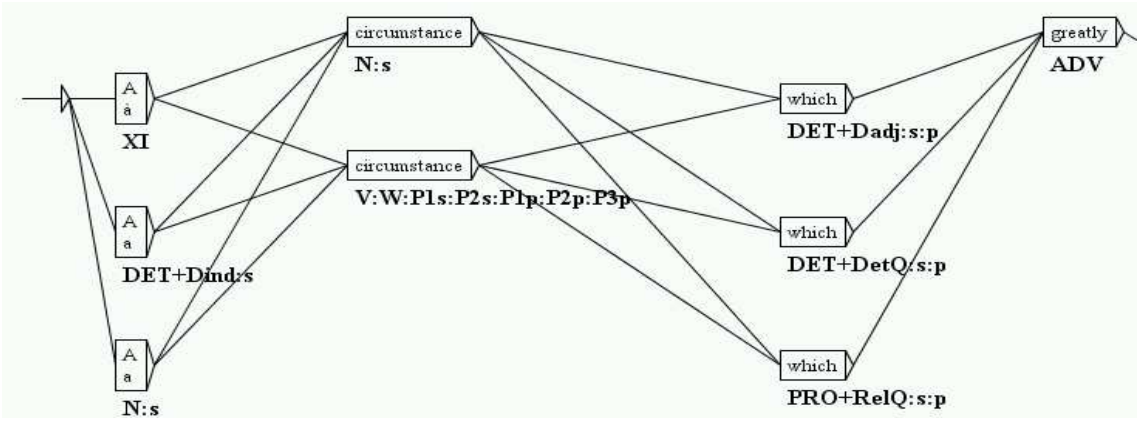

Figure 7.3: Double entry for `which` as a determinator

For each lexical unit of the phrase, Unitex searches for all possible interpretations in the dictionary of the simple words of the text. Afterwards all lexical units that have an interpretation in the dictionary of the composite words of the text are sought. All the combinations of their interpretations constitute the phrase automaton.

NOTE: if the text contains lexical labels (*i.e.* `{aujourd'hui,.ADV}`), these lables are reproduced identically in the automaton whithout trying to decompose the sequences which they represent.

In each box, the 1st line contains the inflected form found in the text, and the 2nd line contains the canonical form if it is different. The other information is coded below the box. (cf. section 7.3.1).

The spaces that separate the lexical units are not copied into the automaton save the spaces inside composite words.

The casing of lexical units is conserved. For example, if the word `Here` is encountered, the capital letter is conserved (cf. figure 7.1). This choice allows to keep this information

during the transition to the text automaton, which could be useful for applications where case is important such as recognition of proper names.

7.2.2 Normalization of ambiguous forms

During construction of the automaton, it is possible to effect a normalization of ambiguous forms by applying a normalization grammar. This grammar has to be called `Norm.fst2` and must be placed in your personal folder, in the subfolder `/Graphs/Normalization` of the desired language. The normalization grammars for ambiguous forms are described in section 6.1.3.

If a sequence of the text is recognized by the normalization grammar, all the interpretations that are described by the grammar are inserted into the text automaton. Figure 7.4 shows the extract of the grammar used for French that makes the ambiguity of the sequence `l'` explicit.

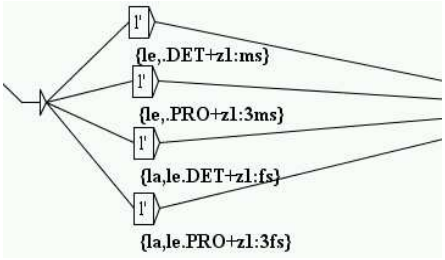

Figure 7.4: Normalization of the sequence `l'`

If this grammar is applied to a french sentence containing the sequence `l'`, a phrase automaton that is similar to the one in figure 7.5 is obtained.

You can see that the four rules for rewriting the sequence `l'` have been applied, which has added four labels to the automaton. These labels are concurrent with the two preexisting paths for the sequence `l'`. The normalization at the time of the construction of the automaton allows to add paths to the automaton but not to erase paths. When the disambiguation functionality will be available it will allow to eliminate the paths that have become superfluous.

7.2.3 Normalization of clitical pronouns in Portuguese

In Portuguese the verbs in future and in conditional can be modified by the insertion of one or two clitical pronouns between the root and the suffix of the verb. For example, the sequence *dir-me-ão* (*they will tell me*), corresponds to the complete verbal form *dirão*, associated to the pronoun *me*. In order to be able to manipulate this rewritten form, it is necessary to introduce it into the text automaton parallely to the original form. Thus, the user could search one or the other form. The figures figures 7.6 and 7.7 show the automaton of a phrase after the normalization of the clitics.

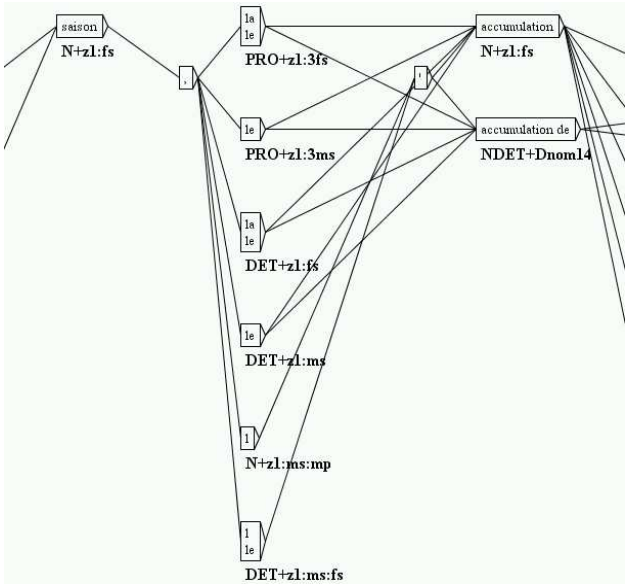

Figure 7.5: Automaton that has been normalized with the grammar of figure 7.4

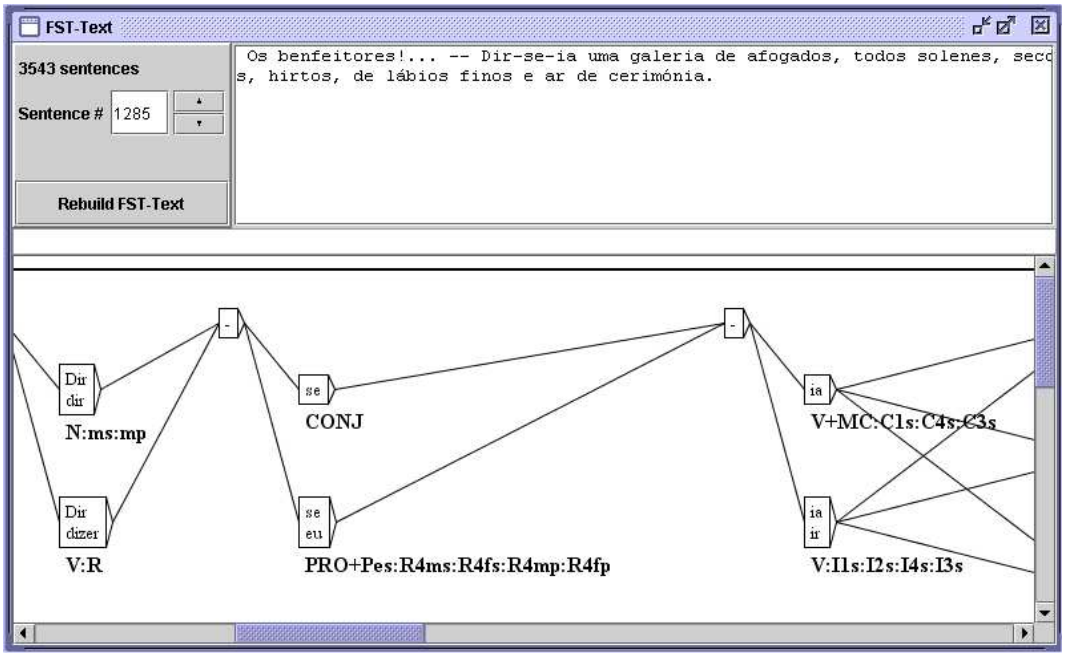

Figure 7.6: Non-normalized phrase automaton

The program `Reconstrucao` allows to construct a normalization grammar for these forms for each text dynamically. The thus produced grammar can then be used for normalizing the text automaton. The configuration window of the automaton construction suggests

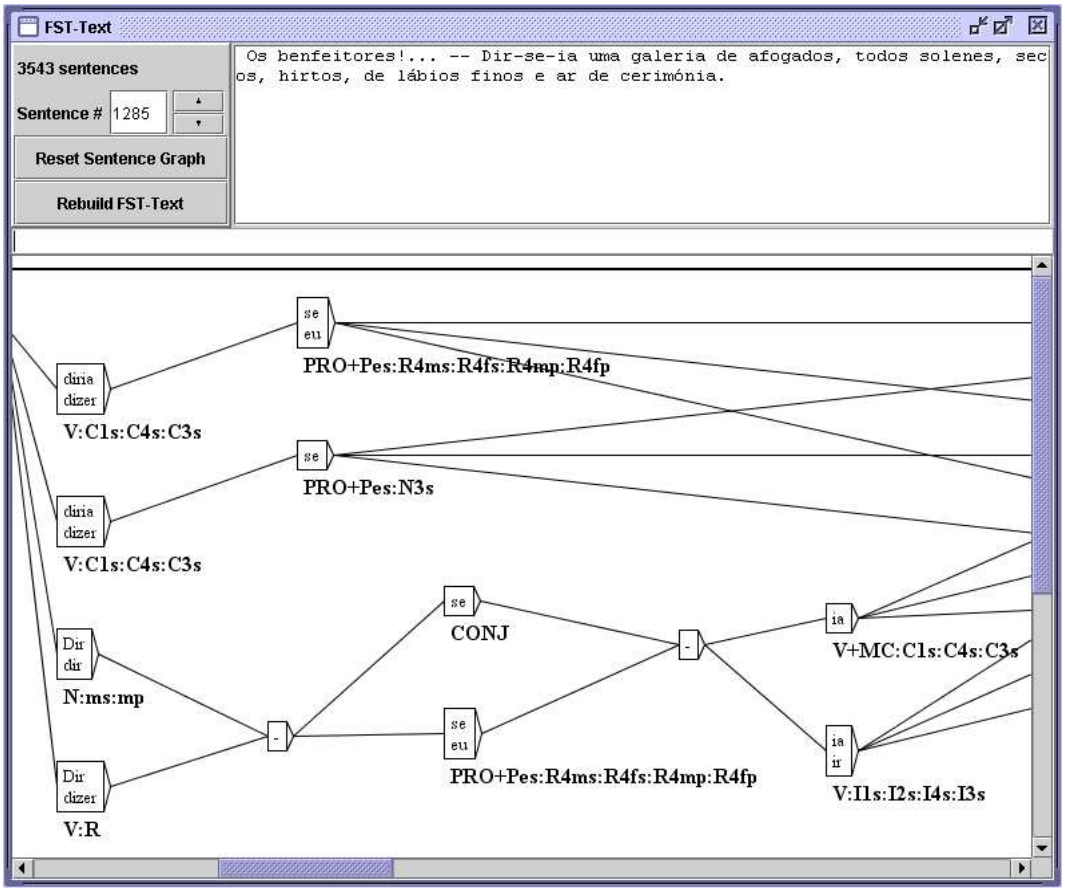

Figure 7.7: Normalized phrase automaton

an option "Build clitic normalization grammar" (cf. figure 7.10). This option automatically starts the construction of the normalization grammar, which is then used to construct the text automaton, if you have selected the option "Apply the Normalization grammar".

7.2.4 Conservation of better paths

It can happen that an unknown word comes paratizing the text automaton by being concurrent with a completely labeled sequence. Thus, in the automaton of figure 7.8, it can be seen that the adverb `aujourd'hui` is concurrent with the unknown word `aujourd`, followed by an apostrophe and the past participle of the verb `huir`.

This phenomenon can also be found in the treatment of certain asian languages like the thaï. When the words are not delimited, there is no other solution than to face all possible combinations, which causes the creation of numerous paths carrying unknown words that are mixed with the labeled paths. Figure 7.9 shows an example of such an automaton of a thaï sentence.

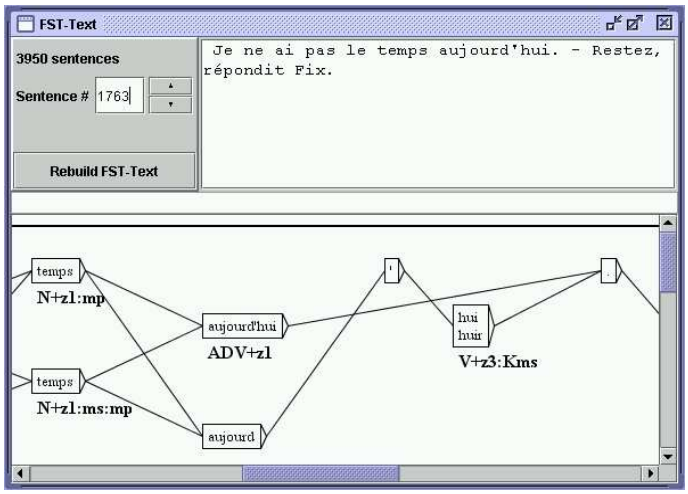

Figure 7.8: Ambiguity due to a sentence conaining an unkown word

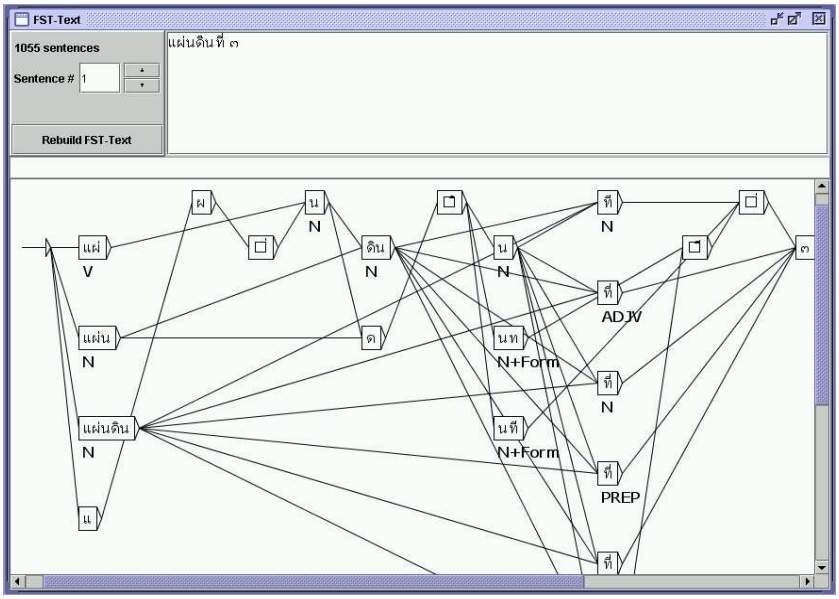

Figure 7.9: Automaton of a thaï phrase

It is possible to suppress parasite paths. You have to select the option "Clean Text FST" in the configuration window for the construction of the text automaton (cf. figure 7.10). This option indicates to the automaton construction program that it should clean up each phrase automaton.

This cleaning is carried out according to the following principle: if several paths are concurrent in the automaton, the program keeps those that contain the least unknown words.

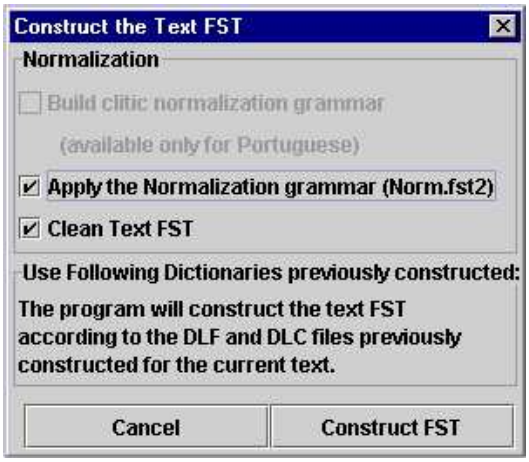
Construct the Text FST
Normalization
Build clitic normalization grammar
(available only for Portuguese)
Apply the Normalization grammar (Norm.fst2)
Clean Text FST
Use Following Dictionaries previously constructed:
The program will construct the text FST according to the DLF and DLC files previously constructed for the current text.
Cancel
Construct FST

Figure 7.10: Configuration of the construction of the text automaton

Figure 7.11 shows the automaton of figure 7.9 after cleaning.

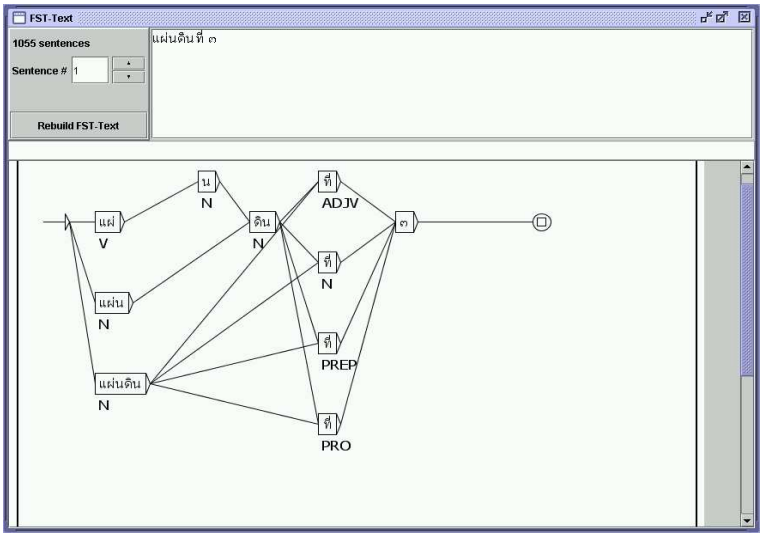

Figure 7.11: Automaton of figure 7.9 after cleaning

7.3 Manipulation

7.3.1 Displaying phrase automata

As we have seen above, the text automaton is in fact the collection of the phrase automata of this text. This structure can be represented using the format `.fst2`, used for representing the compiled grammars. .

Therefore, this format does not allow to directly display the phrase automata. It is therefore necessary to use the a program (`Fst2Grf`) for converting the phrase automaton into a graph that can be displayed. This program is called automatically when you select a phrase in in order to generate the corresponding `.grf` file.

The generated `.grf` files are not interpreted in the same manner as the `.grf` files that represent the graphs that are constructed by the user. In fact, in a normal graph, the lines of a box are separated by the symbol `+`. In the graph of a phrase, each box is either a lexical unit without label or a dictionary entry enclosed by curly brackets. If the box only contains an unlabeled lexical unit, this appears alone in the box. If the box contains a dictionary entry, the inflected form is displayed, followed by the canonical form if it is different. The grammatical and inflectional are displayed below the box as in the transductions.

Figure 7.12 shows the graph obtained by the first phrase of *Ivanhoe*. The words `Ivanhoe`, `Walter` and `Scott` are considered unknown words. The word `by` corresponds to two entires in the dictionary. The word `Sir` corresponds to two dictionary entires as well, but since the canonical form of these entries is `sir`, it is displayed because it differs from the inflected form by a lower-case letter.

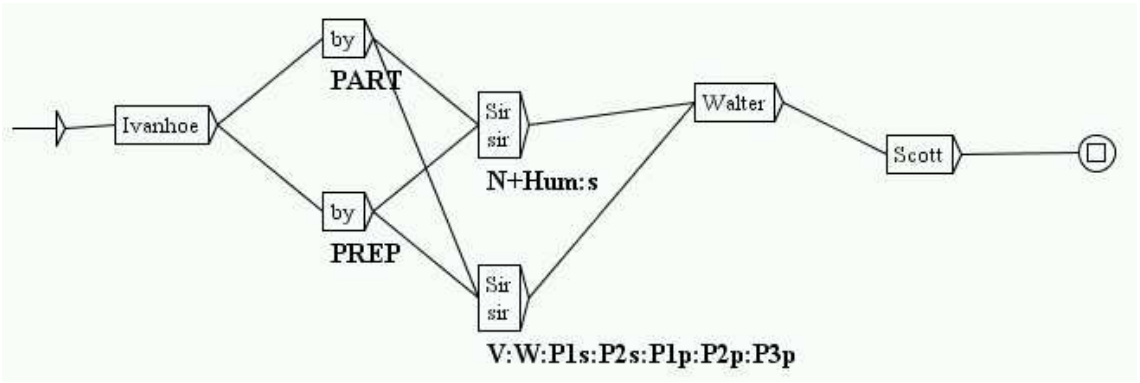

Figure 7.12: Automaton of the first phrase of *Ivanhoe*

7.3.2 Modify the text automaton

It is possible to manually modify the phrase automaton. You can add or erase boxes or transitions. When a graph is modified, it is saved to the text file of the name `sentenceN.grf`, where N represents the number of the phrase.

When you select a phrase, if a modified graph exists for this phrase, this one is displayed. You can then reinitialize the automaton of that phrase by clicking on the botton "Reset Sentence Graph" (cf. figure 7.13).

During the construction of the text automaton all the modified phrase graphs in the text file are erased.

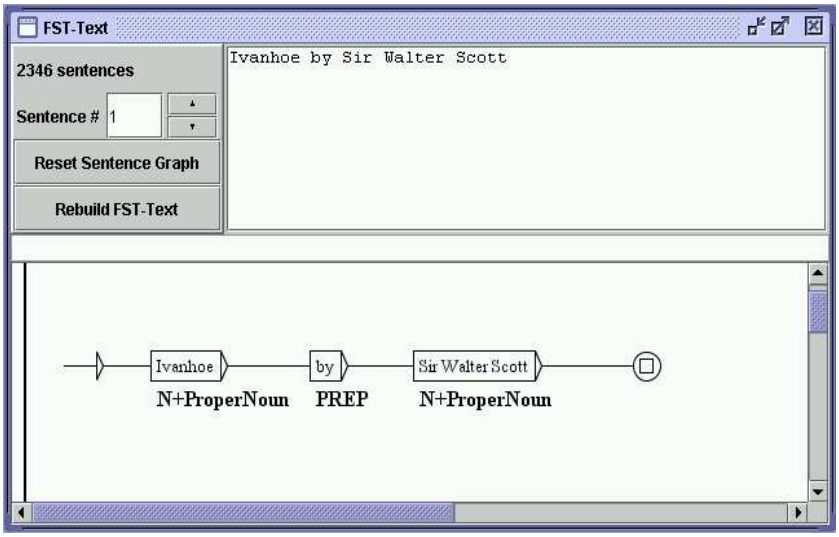

Figure 7.13: Modified phrase automaton

NOTE: you can reconstruct the text automaton while taking into account your manual modifications. In order to do that, click on the button "Rebuild FST-Text". All phrases that have modifications are then replaced in the text automaton with their modified versions. The new text automaton is then automatically reloaded.

7.3.3 Parametres of presentation

the phrase automata are subject to the same presentation options as the graphs. They use the same colors and fonts as well as the antialising effect. In order to configure the appearance of the phrase automata, you have to modify the general configuration by clicking on "Preferences..." in the menu "Info". For further details, refer to the section 5.3.5.

You can also print a phrase automaton by clicking on "Print..." in the menu "FSGraph" or by pressing <Ctrl+P>. Make sure that the printer's page orientation is set to landscape mode. To configure this parameter, click on "Page Setup" in the menu "FSGraph".

Chapter 8

Lexicon Grammar

The tables of the lexicon grammar are a compact way of representing the syntactical properties of the elements of a language. It is possible to automatically construct local grammars from these tables, due to a mechanism of template graphs.

In the first part of the chapter the formalism of the tables is presented. The second part describes the template graphs and mechanism of automatically generating graphs starting from a lexicon grammar table.

8.1 The lexicon grammar tables

The lexicon grammar is a methodology developed by Maurice Gross based on the following principle: every verb has almost unique syntactical properties. Due to this fact, these properties need to be systematically described, since it is impossible to predict the exact behavior of a verb. These descriptions are represented by matrices where the rows correspond to the verbs and the columns to the syntactical properties. The considered properties are formal properties such as the number and nature of allowed complements of the verb and the different transformations the verb can undergo (passivation, nominalisation, extraposition, etc.). The matrixes, mostly called tables, are binary: a + sign at the intersection of a row and a column of a property if the verb has that property, a - sign if not.

This type of description has equally been applied to adjectives, predicative nouns, adverbs, as well as figurative expressions, all in multiple languages.

Figure 8.1 shows an example of a lexicon grammar table. The table concerns verbs that admit a numerical complement.

8.2 Conversion of a table into graphs

8.2.1 Principle of template graphs

The conversion of a table into graphs is carried out by a mechanism of template graphs. The prinicple is the following: a graph that describes the possible constructions is constructed.

Table32NM.xls

1	A N0 = Npc	B N0 = N-hum	C N0 = Nnr	D N0 = V-n	E Aux	F 32NM	G N1 V	H N1 = Nhum	I N1 = N-hum	J N1 = le fait Qu P	K N1 = V-n	L N1 = Dnum Nmes	M Ppv = le	N N-1 V N0	O N0 V Adj	P N0 V Dnum V-n	Q N0 V à N1	R N-1 V N0 (<E>+à) N1	S N1 = V-n	T Exemple
2	-	+	-	-	avoir	accepter	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accepte§vingt personnes
3	-	+	-	-	avoir	accueillir	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accueille§vingt personnes
4	-	+	-	-	avoir	accuser	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§accuse§80 kilos
5	-	+	-	-	avoir	accuser	-	-	+	+	-	-	-	-	-	-	-	-	-	Max§accuse§ses trente ans
6	-	+	-	-	avoir	admettre	-	-	+	-	-	-	+	-	-	-	-	-	-	On§admet§50 personnes dans cette salle
7	-	+	-	-	avoir	affecter	-	-	+	-	-	-	-	-	-	-	-	-	-	Ces cristaux§affectent§une forme géométrique
8	-	+	-	-	avoir	afficher	-	-	+	-	-	+	-	-	-	-	-	-	-	Les valeurs ont§affiché§un repli
9	-	+	-	-	avoir	aimer	-	-	+	+	-	-	+	-	-	-	-	-	-	La plante§aime§l'eau
10	-	+	-	-	avoir	approcher	+	-	+	-	-	+	+	-	-	-	-	-	-	Cette maison§approche§les deux millions
11	-	+	-	-	avoir	arpenter	-	-	-	-	-	+	+	-	-	+	+	+	-	Ce terrain§arpente§30 arpents
12	-	+	-	-	avoir	atteindre	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§atteint§80 kilos
13	+	+	+	-	avoir	avoir	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§a§(une soeur+une voiture+des sous)
14	-	+	-	-	avoir	avoisiner	-	-	+	-	-	+	+	-	-	-	-	-	-	Ce sac§avoisine§les 20 kg.
15	-	+	-	-	avoir	battre	+	-	+	-	-	-	+	-	-	-	-	-	-	La montre§bat§les secondes
16	-	+	-	-	avoir	cacher	-	-	+	-	-	-	-	-	-	-	-	-	-	Son calme§cache§(son+une grande)angoisse
17	-	+	-	-	avoir	caler	-	-	+	-	-	+	+	-	+	-	-	-	-	Ce bateau§cale§80 cm

c32NM

Figure 8.1: Lexicon Grammar Table 32NM 32NM

That graphs refers to the columns of the table in the form of variables. Afterwards, for each line of the table a copy of this graph is constructed where the variables are replaced with the contents of the cell at the intersection of the column that corresponds to the treated line. If a cell of the table contains the + sign, the corresponding variable is replaced by <E>. If the cell contains the - sign, the box containing the corresponding variable is removed, at the same time making the paths through that box unavailable. In all other cases the variable is replaced by the contents of the cell.

8.2.2 Format of the table

The lexicon grammar tables are usually encoded with the aid of a spreadsheet like Microsoft Excel. To make them usable with Unitex, the tables have to be encoded in Unicode text format in accordance with the following convention: the columns need to be separated by a tab and the lines by a newline. To convert a table with Excel, save it in Unicode text format (this operation is only possible with newer versions of Excel). Per default the column separator is a tab, the table therefore needs to be well formatted.

During the generation of the graphs, Unitex skips the first line, considering it to be the headings of the columns. It is therefore necessary to ensure that the headings of the columns occupy exactly one line. If there is no line for the heading, the first line of a table should be ignored, and if there are multiple heading lines, from the second line on they will be interpreted like lines of the table.

8.2.3 The template graphs

The template graphs are the graphs in which the variables appear that refer to the columns of a table of the lexicon grammar. This mechanism is usually used with syntactical graphs, but nothing prevents the construction of template graph for inflection, preprocessing, or for normalization.

The variables that refer to columns are formed with the @ symbol followed by the name of the column in capital letters (the columns are named starting with A).

Example:@C refers to the third column of the table

Whenever a variable needs to be replaced by a + or -, the - sign corresponds to the removal of a path through that variable. It is possible to carry out the contrary operation by putting an exclamation mark in front of the @ symbol. In that case, whenever the variable refers to the + sign, it is replaced by the contents of the cell.

There is also the special variable @% which is replaced by the number of the line in the table. The fact that its value is different for each line allows for its use as the simple characterization of a line. That variable is not affected by an exclamation point to the left of it.

Figure 8.2 shows an example of a template graph designed to be applied to the lexicon grammar table 31H presented in figure 8.3.

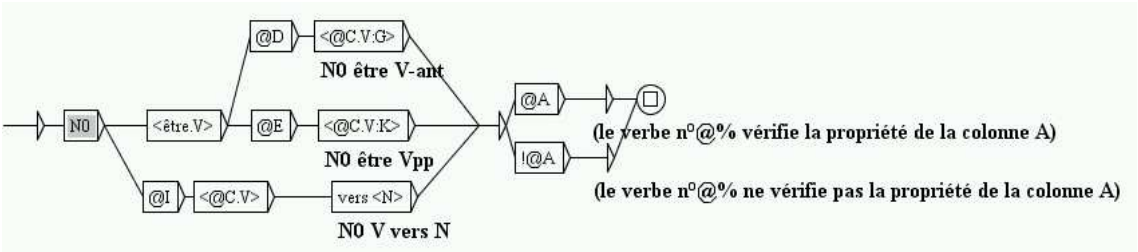

Figure 8.2: Example of template graph

8.2.4 Automatic generation of graphs

In order to be able to generate graphs from a template graph and a table, first of all the table needs to be opened by clicking on "Open..." in the menu "Lexicon-Grammar" (see figure 8.4). The table needs to be in Unicode text format.

The selected table is then opend in a window (see figure figure 8.5).

Table31H.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	N0 = V-n	Aux	31H	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0pc	Nhum V sur ce point	N0 V vers N	il V N0 W	idée Loc esprit	Nhum Loc Nabs	N0 = N-hum	N0 = V-n	Exemple
2	-	avoir	abandonner	-	-	-	-	-	-	-	-	-	-	-	Paul a§abandonné§
3	-	avoir	abuser	-	-	-	-	+	-	-	-	-	-	-	Max§abuse§
4	-	avoir	acquiescer	-	-	-	+	+	-	-	-	-	-	-	Max a§acquiescé§(E+de la tête)
5	-	avoir	adouber	-	-	-	-	-	-	-	-	-	-	-	Paul§adoube§|échecs|
6	-	avoir	agioter	-	-	-	-	-	-	+	-	-	-	-	Max§agiote§sur les changes
7	-	avoir	agoniser	+	-	-	-	-	-	+	-	-	+	-	Max§agonise§
8	-	avoir	archaïser	+	-	-	-	+	-	-	-	-	-	-	Cet auteur§archaïse§volontiers
9	-	avoir	arquer	-	-	-	+	-	+	-	-	-	-	-	Max a§arqué§toute la journée
10	-	être	arriver	-	+	-	-	-	-	+	-	+	-	-	Max est§arrivé§
11	-	avoir	atermoyer	-	-	-	-	+	-	+	-	+	-	-	Max§atermoie§
12	+	avoir	badauder	-	-	-	-	-	+	-	+	-	-	badaud	Max§badaude§

c31H

Figure 8.3: Lexicon grammar table 31H

beta version - Unitex - current language is French
Text DELA FSGraph Lexicon-Grammar Edit Windows Info
Open...
Compile to GRF...
Close

Figure 8.4: Menu "Lexicon-Grammar"

To automatically generate graphs from a template graph, click on "Compile to GRF..." in the menu "Lexicon-Grammar". The window in figure 8.6 shows this

In the frame "Reference Graph (in GRF format)", indicate the name of the template graph that is to be used. In the frame "Resulting GRF grammar", indicate the name of the main graph that will be generated. This main graph is a graph that refers to all graphs that are going to be generated. When launching a search in a text with that graph, all generated graphs are simultaneously applied.

Each of the constructed graphs is controlled by the name of the result graph appended with `_i`, where i represents the number of the line of which it has been generated. For example if the main graph is called TestGraph.grf, the graph generated from the 16þ line is called `TestGraph_0016.grf`.

E:\Table31H.txt

N0 =: V-n	Aux	31H	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0...	Nhum V sur...	N0 V vers N	il V N0 W	i
-	avoir	abando...	-	-	-	-	-	-	-	
-	avoir	abuser	-	-	-	-	+	-	-	
-	avoir	acquie...	-	-	-	+	+	-	-	
-	avoir	adouber	-	-	-	-	-	-	-	
-	avoir	agioter	-	-	-	-	-	-	+	
-	avoir	agoniser	+	-	-	-	-	-	+	
-	avoir	archaiser	+	-	-	-	+	-	-	
-	avoir	arquer	-	-	-	+	-	+	-	
-	être	arriver	-	+	-	-	-	-	+	
-	avoir	atermoyer	-	-	-	-	+	-	+	
+	avoir	badauder	-	-	-	-	-	+	-	
-	avoir	baisser	-	-	-	-	+	-	-	
-	avoir	bambocher	-	-	-	-	-	-	+	
-	avoir	bander	-	-	-	+	-	-	+	
-	avoir	barouder	-	-	-	-	-	-	+	
-	avoir	batifoler	+	-	-	-	-	-	+	
-	avoir	bécher	-	-	-	-	+	-	-	
+	avoir	bétifier	+	-	-	-	+	-	+	
+	avoir	bigler	-	-	+	+	-	+	+	
-	avoir	boiter	-	-	-	+	-	+	+	
-	avoir	boitiller	+	-	-	+	-	+	+	
+	avoir	bouffo...	-	-	-	-	-	-	+	

Figure 8.5: Displaying a table

Figure 8.6: Configuration of the automatic generation of graphs

Figures 8.7 and 8.8 show two graphs generated by applying the template graph of figure 8.2 at table 31H.

Figure 8.9 shows the resulting main graph.

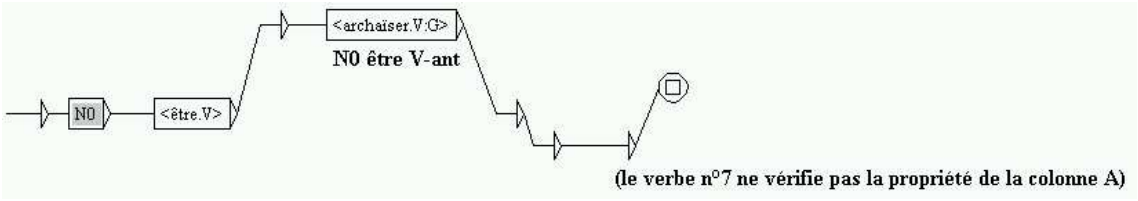

Figure 8.7: Graph generated for the verb `archaïser`

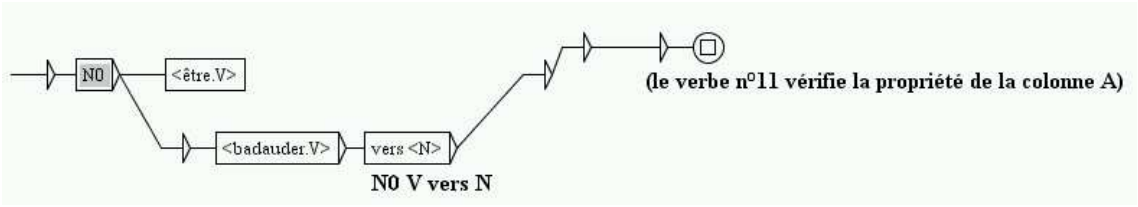

Figure 8.8: Graph generated for the verb `badauder`

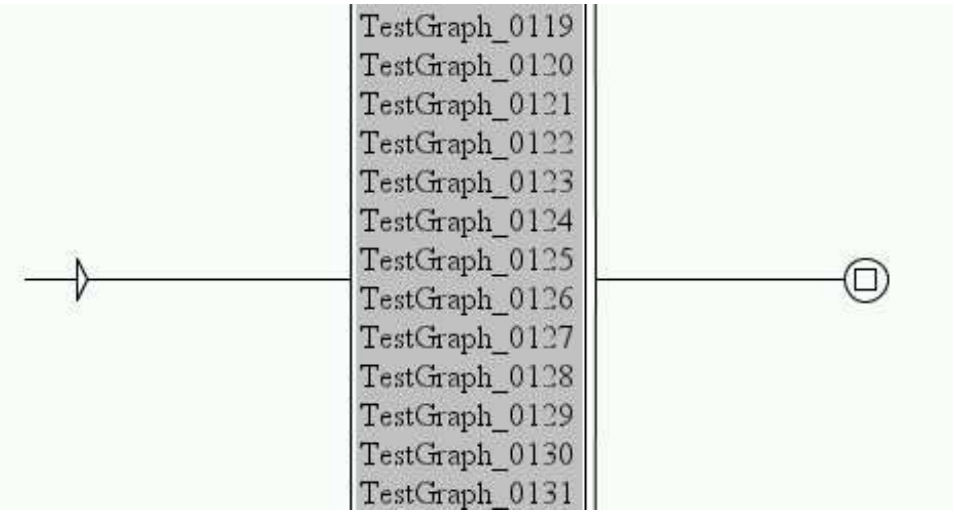

Figure 8.9: Main graph referring to all generated graphs

Chapter 9

Use of external programs

This chapter presents the use of the different programs of which Unitex is composed. These programs, which can be found in the folder `Unitex/App`, are automatically called by the interface. It is possible to see the commands that have been executed by clicking on the menu "Info" on the "Console". It is also possible to see the options of the different programes in the selection of the sub-menu "Help on commands" of the menu "Info".

ATTENTION: multiple programs use the text directory (`my_text_snt`). This directory is created by the graphical interface after the normalization of the text. If you work with the command line, you have to create the directory manually before the execution of the program `Normalize`.

ATTENTION (2): whenever a parameter contains spaces, it needs to be enclosed in quotation marks so it will not be considered as multiple parameters.

IMPORTANT (2): lorsqu'un paramètre contient des espaces, vous devez l'entourer de guillemets pour qu'il ne soit pas considéré comme plusieurs paramètres.

9.1 Asc2Uni

```
Asc2Uni lang text_1 [text_2 ...]
```

This program allows the conversion of ASCII encoded texts into Unicode. The conversion mode is defined by the parameter `lang`. The following values are possible:

- `FRENCH`
- `ENGLISH`
- `GREEK`
- `THAI`
- `CZECH`

- `GERMAN`
- `SPANISH`
- `PORTUGUESE`
- `ITALIAN`
- `NORWEGIAN`
- `LATIN` (encoding used per default for Latin language)
- `windows-1252` Microsoft Windows Codepage 1252 - Latin I (Eastern Europe and USA)
- `windows-1250` Microsoft Windows Codepage 1250 - Central Europe
- `windows-1257` Microsoft Windows Codepage 1257 - Baltic States
- `windows-1251` Microsoft Windows Codepage 1251 - Cyrillic
- `windows-1254` Microsoft Windows Codepage 1254 - Turkish
- `windows-1258` Microsoft Windows Codepage 1258 - Vietnamese
- `iso-8859-1` ISO Character Set 8859-1 - Latin 1 (Eastern Europe and USA)
- `iso-8859-15` ISO Character Set 8859-15 - Latin 9 (Eastern Europe and USA)
- `iso-8859-2` ISO Character Set 8859-2 - Latin 2 (Central and Eastern Europe)
- `iso-8859-3` ISO Character Set 8859-3 - Latin 3 (Southern Europe)
- `iso-8859-4` ISO Character Set 8859-4 - Latin 4 (Northern Europe)
- `iso-8859-5` ISO Character Set 8859-5 - Cyrillic
- `iso-8859-7` ISO Character Set 8859-7 - Greek
- `iso-8859-9` ISO Character Set 8859-9 - Latin 5 (Turkish)
- `iso-8859-10` ISO Character Set 8859-10 - Latin 6 (Nordic)

It is possible to add other encodings by modifying the program, since its source code is distributed with Unitex.

The parameters `text_i` are the names of the files to be converted. The result of the conversion of a file `text_i` is saved in a file named `text_i.uni`.

9.2 CheckDic

```
CheckDic dictionnaire type
```

This program carries out the verification of the format of a dictionary of type DELAS or DELAF. The parameter `dictionnaire` corresponds to the name of the dictionary that is to be verified. The parameter `type` can take the value `DELAS` or `DELAF` depending on the format of the dictionary to be verified.

The program checks the syntax of the lines of the dictionary. It also creates a list of all characters occuring in the inflected and canonical forms of words in the text, the list of gramatical codes and syntax, as well as the list of inflection codes used. The results of the verification are stored in a file called `CHECK_DIC.TXT`.

9.3 Compress

```
Compress dictionary [-flip]
```

This program takes a DELAF dictionary as a parameter and compresses it. The compression of a dictionary `dico.dic` produces two files:

- `dico.bin`: a binary file containing the minimum automaton of the inflected forms of the dictionary
- `dico.inf`: a text file containing the compessed formes allowing the reconstruction of the dictionary lines from the inflected formes contained in the automaton.

For more details on the format of these files, see chapter 10. The optional parameter `-flip` indicates that the inflected and canonical forms should be inversed in the compressed dictionary. This option is used to construct an inverse dictionary which is necessary for the program `Reconstrucao`.

9.4 Concord

```
Concord index font fontsize left right order mode alph [-thai]
```

This program takes an index file of the concordance produced by the program `Locate` and produces a concordance. It is also possible to produce a modified text version taking into account the transductions associated to the occurences. Here the description of the parameters:

- `index`: name of the concordance file. It is necessary to indicate the entire file path, since Unitex uses it to determine for which text the concordance is to be constructed.
- `font`: name of the typeface if the concordance is in HTML format. This value is ignored if the concordance is not in HTML format. nom de la police de caractères à utiliser si la concordance doit être produite au format HTML.

- `fontsize`: size of the typeface if the concordance is in HTML format. This value has to be between 1 and 7. Like the parameter `font`, it is also ignored, if the concordance is not in HTML format.

- `left`: number of characters to the left of the occurrences. In Thai mode, this means the number of non-diacritic characters.

- `right`: number of characters (non-diacritic in Thai mode) to the right of the occurrences. If the occurrence is shorter at this value, the concordance line is completed as if the left context had the same lenght as `right`. If the occurrence has a length longer than the characters defined by `right`, it is nevertheless saved as whole.

- `order`: indicates the mode to use to order the lines of the concordance. The possible values are:

  - `TO`: order in which the occurrences appear in the text;
  - `LC`: left context, occurrence;
  - `LR`: left context, right context;
  - `CL`: occurrence, left context;
  - `CR`: occurrence, right context;
  - `RL`: right context, left context;
  - `RC`: left context, occurrence.
  - `NULL`: does not specify any sorting mode. This option should be used if the text is to be modified instead of constructing a concordance.

  For details on the sorting modes, see section 4.7.2.

- `mode`: indicates in which format the concordance is to be produced. The four possible formats are:

  - `html`: produces a concordance in HTML format encoded in UTF-8;
  - `text`: produces a concordance in Unicode text format;
  - `glossanet`: produces a concordance for GlossaNet in HTML format. The HTML file is encoded in UTF-8;
  - `name_of_file`: indicates to the program that it is supposed to produce a modified version of the text and save it in a file named `name_of_file` (see section 6.4.3).

- `alph`: alphabet file used for sorting. The value NULL indicates the absence of an alphabet file.

- `-thai`: this parameter is optional. It indicates to the program that it is processing a Thai text. This option is necessary to ensure the proper functioning of the program in Thai.

The result of the application of this program is a file called `concord.txt` if the concordance was constructed in text mode, a file called `concord.html` if the mode was `html` or `glossanet`, and a text file with the name defined by the user of the program if the program has constructed a modified version of the text.

In `html` mode, the occurrence is coded like a link. The reference associated to this link is of the form `<a href="X Y Z">`. `X` et `Y` represent the beginning and ending positions of the occurrence of the characters in the file `nom_du_texte.snt`. `Z` represents the number of the sentence in which the occurrence was found.

9.5 Dico

```
Dico texte alphabet dic_1 [dic_2 ...]
```

This program applies dictionaries at a text. The text has to be cut up into lexical units by the program `Tokenize`. The dictionaries need to be compressed with the program `Compress`. `texte` represents the complete file path, without omitting the extension `.snt`. `dic_i` represents the file path of a dictionary. The dictionary must have the extensin `.bin`. It is possible to give priorities to the dictionaries. For details see section3.6.1.

The program `Dico` produces the following four files, and saves them in the directory of the text:

- `dlf`: dictionary of simple words in the text;
- `dlc`: dictionary of composed words in the text;
- `err`: list of unknown words in the text;
- `stat_dic.n`: file containing the number of simple words, the number of composed words, and the number of unknown words in the text.

NOTE: the files `dlf`, `dlc` and `err` are not sorted. Use the program `SortTXT` to sort them.

9.6 Extract

```
Extract yes/no texte concordance resultat
```

This program takes a text and a concordance as parameters. If the first parameter is `yes`, the program extracts all sentences from the text that have at least one occurrance from the concordance. If the parameter is `no`, the program extracts all sentences that do not contain any occurrences from the concordance. The parameter `text` represents the complete path of the text file, without omitting the extension .snt. The parameter `concordance` represents the complete path of the concordance file, without omitting the extension .ind.

The parameter `result` represents the name of the file in which the extracted sentences are to be saved.

The result file is a text file that contains all extracted sentences, one sentence per line.

9.7 Flatten

```
Flatten fst2 type [depth]
```

This program takes an ordinary grammar as its parameter, and tries to transform it into a final state transducer. The parameter `fst2` indicates the grammar to transform. The parameter `type` indicates which kind of grammar the result grammar should be. If this parameter is `FST`, the grammar is "unfolded" to maximum depth and is truncated if there are calls to sub-graphs. The result is a grammar in `.fst2` format that does only contain a single final state transducer.

If the parameter is `RTN`, the calls of sub-graphs that could remain after the transformation are left as they are. The result is therefore a final state transducer in the favorable case, and an optimized grammar strictly equivalent to the original grammar if not. The optional parameter `depth` idicates the maximum depth of overlapping of the sub-graphs that are generated by the program. The default value is 10.

9.8 Fst2Grf

```
Fst2Grf text_automaton sentence
```

This program extracts an automaton of a sentence in `.grf` format from the automaton of a text. The parameter `text_automaton` represents the complete path of the automaton file of the text from which a sentence is to be extracted. This file is called `text.fst2` and is stored in the directory of the text. The parameter `sentence` indicates the number of sentences to extract.

The program produces the following two files and saves them in the directory of the text:

- `cursentence.grf`: graph representing the automaton of the sentence
- `cursentence.txt`: text file containing the sentence.

9.9 Fst2Txt

```
Fst2Txt text fst2 alphabet mode [-char_by_char]
```

This program applies a transducer at a text at the preprocessing stage, when the text has not been cut into lexical units yet. The parameters of the program are the following:

- `text`: the text file to modify, with the extension `.snt`;

- `fst2`: the transducer to apply;
- `alphabet`: the alphabet file of the language of the text;
- `mode`: the application mode of the transducer. The two possible modes are `-merge` and `-replace`;
- `-char_by_char`: this optional parameter permits the application of the transducer in "character by character" mode. This option is used for texts in Asian languages.

This program modifies the text file given as a parameter.

9.10 Grf2Fst2

```
Grf2Fst2 graph [y/n]
```

This program compiles a grammar into a file `.fst2` (for more details see section 6.2). The parameter `graphe` denotes the complete path of the main graph of the grammar, without omitting the extension `.grf`. The second parameter is optional. It indicates to the program whether the grammar needs to be checked for errors or not. Per default, the program carries out this error check.

The result is a file that carries the same name as the graph passed to the program as a parameter, but with the extension `.fst2`. This file is saved in the same folder as `graph`.

9.11 Inflect

```
Inflect delas result rep
```

This program carries out the automatic inflection of a DELAS dictionary. The parameter `delas` indicates the name of the dictionary to inflect. The parameter `result` indicates the name of the dictionary to be generated. The parameter `rep` indicates the complete file path of the directory in which the inflection transducers are that the `delas` dictionary refers to.

The result of the inflection is a DELAF dictionary saved under the name indicated by the parameter `result`.

9.12 Locate

```
Locate text fst2 alphabet s/l/a i/m/r n [-thai] [-space]
```

This program applies a grammar to a text and costructs an index file of the found occurrences. The following are its parameters:

- `text`: complete path of the text file, without omitting the extension `.snt`;

- `fst2`: complete path of the grammar, without omitting the extension `.fst2`;
- `alphabet`: complete path of the alphabet file;
- `s/l/a`: parameter indicating whether the search should be carried out in mode *shortest matches* (`s`), *longest matches* (`l`) ou *all matches* (`a`);
- `i/m/r`: parameter indicating the application mode of the transductions: mode MERGE (`m`) or mode REPLACE. `i` indicates that the program should not take into account transductions;
- `n`: parameter indicating how many occurences to search for; The value `all` indicates that all occurrences need to be extracted;
- `-thai`: optional parameter necessary for searching a Thai text;
- `-space`: optional parameter indicating that the search should be performed beyond spaces. This parameter should only be used to carry out morphological searches.

This program saves the references to the found occurrences in a file called `concord.ind`. The number of occurrences, the number of converted units due to those occurrences, as well as the percentage of recognized units within the text are saved in a file called `concord.n`. These two files are stored in the directory of the text.

9.13 MergeTextAutomaton

```
MergeTextAutomaton automaton
```

This program reconstructs the text automaton `automaton` taking into account the modifation manually conducted. In addition to that, if the program finds a file `sentenceN.grf` in the same directory as `automaton`, it replaces the automaton of sentence `N` with the one represented by `sentenceN.grf`. The file `automaton` is replaced by the new text automaton. The old text automaton is backed up in a file called `text.fst2.bck`.

9.14 Normalize

```
Normalize txt
```

This program carries out a normalization of text separators. The separators are space, tab, and newline. Every sequence of separators that contain at least one newline is replaced by a unique newline. All other sequences of separators are replaced by a single space.

This program also verifies the syntax of the lexical lables presented in the text. All sequences in curly brackets is either the sentence delimiter `{S}`, or a valid line of DELAF (`{aujourd'hui,.ADV}`). If the program finds curly brackets that are used differently, it gives a warning and replaces them by square brackets (`[` and `]`). The parameter 'txt' represents the complete path of the text file. The program creates a modified version of the text that is saved in a file with the extension `.snt`.

9.15 PolyLex

```
PolyLex lang alph dic list out [info]
```

This program takes a file with unknown words `list` and tries to analyse each of the words a compound obtained by combining simple words. The words that have at least one analysis are removed from the file of unknown words and the dictionary lines that correspond to the analysis are appended to the file `out`. The parameter `lang` determines the language to use. The two possible values are `GERMAN` and `NORWEGIAN`. The parameter `alph` represents the alphabet file to use. The parameter `dic` designates which dictinary to consult for the analysis. The parameter `out` designates the file in which the produced dictionary lines are to be printed; if that file already exists, the produced lines are appended at the end of the file. The optional parameter `info` designates a text file in which the information about the analysis has been produced.

9.16 Reconstrucao

```
Reconstrucao alph concord dic reverse_dic pro res
```

This program generates a normalization grammar designed to be applied before the construction of an automaton for a Portugese text. The parameter `alph` designates the alphabet file to use. The file `concord` represents a concordance which has to be produced by the application in MERGE mode to the considered thext of a grammar that extracts all forms to normalize. This grammar is called `V-Pro-Suf`, and is stored in the directory `/Portuguese/Graphs/Normalization`. The parameter `dic` designates which dictinary to use to find the canonical forms that are associated to the roots of the verbs. `reverse_dic` designates the inverse dictionary to use to fid the forms in future and conditional starting from canonical forms. These two dictionaries have to be in `.bin` format, and `reverse_dic` has to be obtained by compressing the dictionary of verbs in future and conditional with the parameter `-flip` (see section 9.3). The parameter `pro` designates the grammar of reentry of the pronoms to use. `res` designates the file `.grf` into which the normalization rules are to produce.

9.17 Reg2Grf

```
Reg2Grf fic
```

This program constructs a file `.grf corresponding to the regular expression in file \ve`
The parameter `fic` represents the complete path to the file containing the regular expression. This file needs to be a Unicode text file. The program takes into account all characters up to the first newline. The result file is called `regexp.grf` and is saved in the same directory as `fic`.

9.18 SortTxt

```
SortTxt text [OPTIONS]
```

This program carries out a lexicographical sorting of the lines of the file `text`. `text` represents the complete path of the file to sort. The possible options are:

- `-y`: delete double;
- `-n`: conserve doubles;
- `-r`: sort in descending order;
- `-o fic`: sort using the alphabet of the order defined by the file `fi`. If this parameter is missing, the sorting is done according to the order of the Unicode characters;
- `-l fic`: backup the number of lines of the result file in the file `fic`;
- `-thai`: option for sorting a Thai text.

The sort operation modifies the file `text`. By default, the sorting is effected in the order of the Unicode characters, removing doubles.

9.19 Table2Grf

```
Table2Grf table grf result.grf
```

This program automatically generates graphs from a lexicon-grammar `table` and the template graph `grf`. The name of the produced main graph of the grammar is `result.grf`. The names of the produced sub-graphs are of the form `result_i.grf`.

9.20 TextAutomaton2Mft

```
TextAutomaton2Mft text.fst2
```

This program takes a text automaton `text.fst2` as a parameter and constructs the equivalent in the `.mft` format of Intex. The produced file is called `text.mft` and is encoded in Unicode.

9.21 Tokenize

```
Tokenize text alphabet [-char_by_char]
```

This program cuts the text into lexical units. The parameter `texte` represents the complete path of the text file, without omitting the extension `.snt`. The parameter `alphabet` represents the complete path of the alphabet definition file of the language of the text. The

optional parameter `-char_by_char` indicates whether the program is applied character by character, with the exception of the sentence separator `{S}` which is considered to be a single unit. Without this parameter set, the program considers a unit to be either a sequence of letters (the letters are defined by the file `alphabet`), or a character which is not a letter, or the sentence separator `{S}`, or a lexical label (`{aujourd'hui,.ADV}`).

The program codes each unit as a whole. The list of units is saved in a text file called `tokens.txt`. The sequence of codes representing the units now allows the coding of the text. This sequence is saved in a binary file named `text.cod`. The program also produces the following four files:

- `tok_by_freq.txt`: text file containing the units ordered by frequency;
- `tok_by_alph.txt`: text file containing the units ordered alphabetically;
- `stats.n`: text file containing information on the number of sentence separators, the number of units, the number of simple words and the number of numbers;
- `enter.pos`: binary file containing the list of newline positions in the text. The coded representation of the text does not contain newlines, but spaces. Since a newline counts for two characters and the space for a single one, it is necessary to know where there are newlines in the text if the posistions of the calcualted occurrences by the program `Locate` are to by synchronized with the text file. For this the file `enter.pos` is used by the program `Concord`. Thanks to this, when clicking on an occurrence in a concordance, it is correctly selected in the text.File binaire contenant la liste des positions des retours à la ligne dans le texte.

All produced files are saved in the directory of the text.

9.22 Txt2Fst2

```
Txt2Fst2 text alphabet [-clean] [norm]
```

This program constructs an automaton of a text. The parameter `text` represents the complete path of a text file without omitting the extesion `.snt`. The parameter `alphabet` represents the complete path of the alphabet file of the language of the text. The optional parameter `-clean` indicates whether the principle of conservation of the best paths (see section 7.2.4) should be applied. If the parameter `norm` is specified, it is interpreted as the name of a normalization grammar that is to by applied to the text automaton.

If the text is separated into sentences, the program constructs an automaton for each sentence. If this is not the case, the program arbitrarily cuts the text into sequences of 2000 lexical units and produces an automaton for each of these sequences.

The result is a file called `text.fst2` which is saved in the directory of the text.

9.23 Uni2Asc

```
Uni2Asc lang text_1 [text_2 ...]
```

This program allows the conversion of Unicode text files into ASCII. The conversion modes used are defined by the parameter `lang`. The possible values are the same as for the program `Asc2Uni`, , but there is an additinal mode with the value `UTF-8`, which indicates that the Unicode Little-Endian files should be converted into UTF-8.

The parameters `text_i` are thenames of the files to convert. The result of the conversion of a file `text_i` is saved in a file called `text_i.ascii`.

Chapter 10

File formats

This chapter presents the formats of files read or generated by Unitex. The formats of the DELAS and DELAF dictionaries have already been presented in sections 3.1.1 and 3.1.2.

NOTE: in this chapter the symbol ¶ represents the newline symbol. Unless otherwise indicated, all text files described in this chapter are encoded in Unicode Little-Endian.

10.1 Unicode Little-Endian encoding

All text files processed by Unitex have to be encoded in Unicode Little-Endian. This encoding allows the representation of 65536 characters by coding each of them in 2 bytes. In Little-Endian, the bytes are in lo-byte hi-byte order. If this order is reversed, we speak of Big-Endian. A text file encoded in Unicode Little-Endian starts with the special character with the hexadecimal value `FEFF`. The newline symbols have to be encoded by the two characters `000D` and `000A`.

Consider the following text:

```
Unitex¶
β-version¶
```

Here its representation in Unicode Little-Endian:

en-tête	U	n	i	t	e	x	¶	β
FFFE	5500	6E00	6900	7400	6500	7800	0D000A00	B203
-	v	e	r	s	i	o	n	¶
2D00	7600	6500	7200	7300	6900	6F00	6E00	0D000A00

Table 10.1: Hexadecimal representation of a Unicode text

The hi-bytes and lo-bytes have been reversed, which explains why the start character is encoded as `FFFE` in stead of `FEFF`, and `000D` and `OO0A` are `0D00` and `0A00` respectively.

10.2 Alphabet files

There are two kinds of alphabet files: a file which defines the characters of a language, and a file that indicates the sorting preferences. The first is designed under the name *alphabet*, the second under the name *sorted alphabet*.

10.2.1 Alphabet

The alphabet file is a text file that describes all characters of a language, as well as the correspondances between capitalized and non-capitalized letters. This file is called `Alphabet.txt` and is found in the root of the directory of a language. Its presence is obligatory for Unitex to function.

Example: the English alphabet file has to be in the directory `.../English/`

Each line of the alphabet file must have one of the following three forms, followed by a newline symbol:

- `#가힣` : a hash symbol followed by two characters X and Y which indicate that all characters between X and Y are letters. All these characters are considered to be in non-capitalized and capitalized form at the same time. This method is used to define the alphabets of Asian languages like Korean, Chinese or Japanese where there is no destinction between upper- and lower-case, and where the number of characters makes a complete enumeration where tedious;
- `Ëë` : two characters X and Y indicate that X and Y are letters and that X is equivalent in its capitalized and non-capitalized form.
- `ঌ`: a unique character X defines X as a letter in capitalized and non-capitalized form. This form is used to define an Asian punctuation mark.

For certain languages like French, it is possible that a lower-case letter corresponds multiple upper-case letters, like for example `é`, which can have the upper-case form `E` or `É`. To express this, it suffices to use multiple lines. The inverse is equally true: a capitalized letter can correspond to multiple lower-case letters. Thus, the `E` can be the capitalization of `e`, `é`, `è`, `ê` or `ë`. Here an abstract of the French alphabet file which defines the different letters `e`:

```
Ee¶
Eé¶
Éé¶
Eè¶
Èè¶
Eê¶
Êê¶
Eë¶
Ëë¶
```

10.2.2 Sorted alphabet

The sorted alphabet text file defines the sorting priorities of the letters of a language with which to sort with the program `SortTxt`. Each line of that file defines a group of letters. If a group of letters A is defined before a group of letters B, every letter of group A is inferior to every letter in group B.

The letters of a group are only distinguished if necessary. For example if the group of letters `eéèêë` has been defined, the word `ébahi` should be considered 'smaller' than `estuaire`, and also 'smaller' than `été`. Since the letters that follow `e` and `é` allow a classification of the words, it is not necessary to compare the letters `e` and `é` since they are of the same group. On the other hand, if the words `chantés` and `chantes` are to be sorted, `chantes` should be considered as 'smaller'. It is therefore necessary to compare the letters `e` and `é` to distinguish these words. Since the letter `e` appears first in the group `eéèêë`, it is considered to be 'smaller' than `chantés`. The word `chantes should therefore be considered to b`

The sorted alphabet file allows the definition of equivalent characters. It is therefore possible to ignore the different accents as well as capitalization. For example, if the letters `b`, `c`, and `d` are to be ordered without considering capitalization and the cedilla, it is possible to write the following lines:

```
Bb¶
CcÇç¶
Dd¶
```

This file is optional. If no sorted alphabet file is specified, the program `SortTxt` creates a sorting in the order of the Unicode encoding.

10.3 Graphs

This section presents the two graph formats: the graphic format `.grf` and the compiled format `.fst2`.

10.3.1 Format .grf

A `.grf` file is a text file that contains presentation information in addition to information representing the contents of the boxes and the transitions of the graph. A `.grf` file begins with the following lines:

```
#Unigraph¶
SIZE 1313 950¶
FONT Times New Roman:  12¶
OFONT Times New Roman:B 12¶
BCOLOR 16777215¶
FCOLOR 0¶
ACOLOR 12632256¶
SCOLOR 16711680¶
```

```
CCOLOR 255¶
DBOXES y¶
DFRAME y¶
DDATE y¶
DFILE y¶
DDIR y¶
DRIG n¶
DRST n¶
FITS 100¶
PORIENT L¶
#¶
```

The first line `#Unigraph` is a comment line. The following lines define the parameter values of the graph presentation:

- `SIZE x y` : defines the width `x` and the hight `y` of a graph in pixesl;
- `FONT name:xyz` : defines the font used for displaying the contents of the boxes. `name` represents the name of the mode. `x` indicates if the text should be in bold face or not. If `x` is `B`, it indicates that it should be bold. For non-bold face, `x` should be a space. In the same way, `y` has the value `I` if the text should be italic, a space if not. `z` represents the size of the text;
- `OFONT name:xyz` : defines the mode used for displaying the transductions. The parameters `name`, `x`, `y`, and `z` are defined in the same way as `FONT`;
- `BCOLOR x` : defines the background color of the graph. 'x' represents the color in RGB format;;
- `FCOLOR x` : defines the desing color of the graph. 'x' represents the color in RGB format;
- `ACOLOR x` : defines the color used for designing the lins of the boxes that correspond at the calls of sub-graphs. `x` represents the color in RGB format;
- `SCOLOR x` : defines the color used for writing in the comment box (for example the boxes that are not linked up with any others). `x` represents the color in RGB format;
- `CCOLOR x` : defines the color used for designing the selected boxes. `x` represents the color in RGB format;
- `DBOXES x` : this line is ignored by Unitex. It is conserved to ensure the compatibility with Intex graphs;
- `DFRAME x` : designs or not a frame around the graph if `x` is `y`, not if it is `n`;
- `DDATE x` : puts the date at the bottom of the graph if `x` is `y`, not if it is `n`;
- `DFILE x` : puts the name of the file at the bottom of the graph depending on whether `x` is `y` or `n`;

- `DDIR x` :
- `DRIG x` : designes the graph from right to left or left to right depending on whether `x` is `y` or `n`;
- `DRST x` : this line is ignored by Unitex. It isconserved to ensure the compatibility with Intex graphs;
- `FITS x` : this line is ignored by Unitex. It isconserved to ensure the compatibility with Intex graphs;
- `PORIENT x` : this line is ignored by Unitex. It isconserved to ensure the compatibility with Intex graphs;
- `#` : this line is ignored by Unitex. It serves to indicate the end of the header information.

The following lines give the contents and the position of the boxes in the graph. The following lines correspond to a graph recognizing a number:

```
3¶
"<E>" 84 248 1 2 ¶
"" 272 248 0 ¶
s"1+2+3+4+5+6+7+8+9+0" 172 248 1 1 ¶
```

The first line indicates the number of boxes in the graph, immediately followed by a newline. This number can not be lower than 2, since a graph always has an initial and a final state.

The following lignes define the boxes of the graph. The boxes are numbered starting at 0. By convention, state 0 is the initial state and state 1 is the final state. The contents of the final state is always empty.

Each box in the graph is defined by a line that has the following format:

contents X Y N transitions ¶

contents is a sequence of characters enclosed in quotation marks that represents the contens of the box. This sequence can sometimes be preceeded by an `s` if the graph is imported from Intex; this character is then ignored by Unitex. The contents of the sequence is the text that has been entered in the editing line for graphs. The following table shows the encoding of two special sequences that are not encoded in the same way as they are entered into the files `.grf`:

Le contenu de la chaîne est le texte qui a été entré dans le contrôle de texte de l'éditeur de graphes. Le tableau suivant donne le codage des deux séquences spéciales qui ne sont pas codées telles quelles dans les Files `.grf`:

Sequence in the graph editor	Sequence in the file `.grf`
"	\"
\"	\\\"

Table 10.2: Encoding of special sequences

NOTE: The characters between < and > or between { and } are not being interpreted. Thus the character + in the sequence `"le <A+Conc>"` is not interpreted like a line separator, since the pattern `<A+Conc>` is interpreted with priority.

X and *Y* represent the coordinates of the box in pixels. Figure 10.1 shows how these coordinates are interpreted by Unitex.

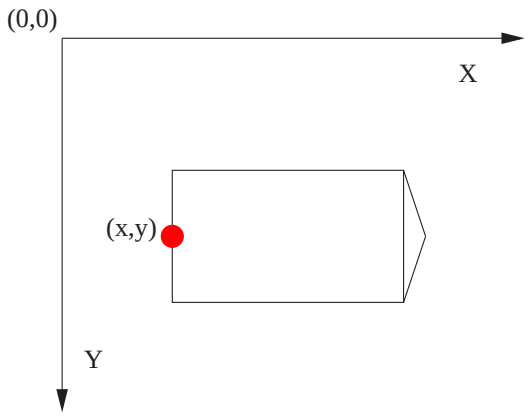

Figure 10.1: Interpretation of the coordinates of boxes

N represents the number of transitions that leave the box. This number is always 0 for the final state.

The transitions are defined by the numbers of boxes at which they point.

Every line of the box definition ends with a newline.

10.3.2 Format .fst2

An `.fst2` file is a text file that describes a set of graphs. Here an example of an `.fst2` file:

```
0000000002¶
-1 GN¶
: 1 1 ¶
: 2 2 -2 2 ¶
```

```
: 3 3 ¶
t ¶
f ¶
-2 Adj¶
: 6 1 5 1 4 1 ¶
t ¶
f ¶
%<E>¶
%le/DET¶
%<A>/ADJ¶
%<N>¶
%beau¶
@joli¶
%petit¶
f¶
```

The first line represents the number of graphs that are encoded in the file. The beginning of each graph is identified by a line that indicates the number and the name of the graph (`-1 GN` and `-2 Adj` in the file below).

The following lines describe the states of the graph. If the state is final, the line starts with the character `t` and with the character `:` if not. For each state, the list of transitions is a possibly empty sequence of entity pairs:

- the first entity indicates the number of the label where the sub-graph corresponds to the transition. The labels are numbered starting at 0. The sub-graphs are represented by negative entities, which explains why the numbers preceeding the names of the graphs are negativ;
- the second entity represents the number of the result state after the transition. In each graph, the states are numbered starting at 0. By convention the state 0 of a graph is its initial state.

Each definition line of a state terminates with a space. The end of each graph is marked by a line containing an `f` followed by a space.

The labels ar defined after the last graph. If the line begins with the `@` character, the contents of the lable is to be searched in its capitalization variant. This information is not used if the lable is a word. If the line starts with a `%`, the capitalization variants are authorized. If a lable carries a transduction, the input and output sequences are separated by the `/` character (example: `the/DET`). By convention, the first label is always the empty word (`<E>`), even if that lable is never used for any transition.

The end of the file is indicated by a line containing the character `f` followed by a newline.

10.4 Texts

This section presents the different files used to represent texts.

10.4.1 .txt files

The `txt+ files are text files encoded in Unicode Little-Endian` These files should not contain any opening or closing braces, except for those used to mark a sentence separator (`{S}`) or a valid lexial label (`{aujourd'hui,.ADV}`). The newline needs to be encoded with the two special characters with the hexadecimal values `000D` and `000A`.

10.4.2 .snt Files

The `.snt` files are `.txt` files that have been processed by Unitex. These files should not contain any tabs. They should also not contain multiple consecutive spaces or newlines. The only allowed braces in the `.snt files are those of the sentence separator \verb`S+ and those of lexical labels (`{aujourd'hui,.ADV}`).

10.4.3 File text.cod

The file `text.cod` is a binary file containing a sequence of entities that represent the text. Each entity `i` reflects the token with index `i` in the file `tokens.txt`. These entities are encoded in for bytes.

NOTE: The tokens are numbered starting at 0.

10.4.4 The file tokens.txt

The file `tokens.txt` is a text file that contains the list of all lexical units of the text. The first line of this file indicates the number of units found in the file. The units are separated by a newline. Whenever a sequence is found in the text with capitalization variants, each variant is encoded as a distinct unit.

NOTE: the newlines that might be in the file `.snt` are encoded like spaces. Therefore there is never a unit encoding the newline.

10.4.5 The files tok_by_alph.txt and tok_by_freq.txt

These two files are text files that contain the list of lexical units sorted alphabetically or by frequence.

In the `tok_by_alph.txt file`, each line is composed by a unit, followed by a tab and the number of occurrences of the unit within the text.

The lines of the `tok_by_freq.txt` file are formed after the same principle, but the number of occurrences occurs after the tab and the unit.

10.4.6 The file enter.pos

This file is a binary file containing the list of positions of the newline symbol in the file `.snt`. Each position is the index in the file `text.cod` where a newline has been replaced by a space. These positions are entities that are encoded in 4 bytes.

10.5 Text Automaton

10.5.1 The file text.fst2

The file `text.fst2` is a special `.fst2` file that represents the text automaton. In that file, each sub-graph represents a sentence automaton. The areas reserved for the names of the sub-graphs are used to store the sentences from which the sentence automata have been constructed.

With the exception of the first lable which is always the empty word (`<E>`), the lables have to be either lexical units or entries from DELAF in braces.

Example: Here the file that corresponds to the text *Il mange une pomme de terre.*

```
0000000001¶
-1 Il mange une pomme de terre. ¶
: 1 1 ¶
: 2 2 ¶
: 3 3 4 3 ¶
: 5 4 6 4 7 4 ¶
: 8 5 9 5 10 5 ¶
: 11 6 12 6 ¶
: 13 7 ¶
t ¶
f ¶
%<E>¶
%{Il,il.PRO+z1:3ms}¶
%{mange,manger.V+z1:P1s:P3s:S1s:S3s:Y2s}¶
%{une,une.N+z1:fs}¶
%{une,un.DET+z1:fs}¶
%{pomme,pomme.A+z1:ms:fs:mp:fp}¶
%{pomme,pomme.N+z1:fs}¶
%{pomme,pommer.V+z3:P1s:P3s:S1s:S3s:Y2s}¶
%{de,de.DET+z1}¶
%{de,de.PREP+z1}¶
%{terre,terre.N+z1:fs}¶
%{terre,terrer.V+z1:P1s:P3s:S1s:S3s:Y2s}¶
%.¶
f¶
```

10.5.2 The file cursentence.grf

The file `cursentence.grf` is generated by Unitex during the display of a sentence automaton. The program `FST2Grf` constructs a file `.grf` from the file `text.fst2` that represents a sentence automaton.

10.5.3 The file sentenceN.grf

Whenever the user modifies a sentence automaton, that automaton is saved under the name `sentenceN.grf`, where `N` represents the number of the sentence.

10.5.4 The file cursentence.txt

During the extraction of the sentence automaton, the text of the sentence is saved in the file called `cursentence.txt`. That file is used by Unitex to display the text of the sentence under the automaton. That file contains the text of the sentence, followed by a newline.

10.6 Concordances

10.6.1 The file concord.ind

The file `concord.ind` is the index of the occurrences found by the program `Locate` during the application of a grammar. It is a text file that contains the starting and end position of each occurrence, possibly accompanied by a sequence of letters if the concordance has been obtained by taking into account the possible transductions of the grammar. Here an example of a file:

```
#M¶
3036 3040 le[ADJ= petit] salon¶
3071 3075 Le nouveau domestique¶
5600 5604 le jeune Lord¶
6052 6056 le second étage¶
6123 6127 le premier étage¶
6181 6185 le même instant¶
6461 6465 le méthodique gentleman¶
7468 7472 le grand salon¶
7520 7524 le laborieux dépliage¶
7675 7679 le grand salon¶
8590 8594 le fait plus¶
10990 10994 le mauvais temps¶
13719 13723 le brave garçon¶
13896 13900 le modeste sac¶
15063 15067 le même compartiment¶
```

The first line indicates in which transduction mode the concordance has been constructed. The three possible values are:

- `#I` : the transductions have been ignored;
- `#M` : the transductions have been insorted into the recognize sequences (MERGE mode);
- `#R` : the transductions have replaced the recognized sequences (REPLACE mode)).

Each occurrence is described in one line. The lines start with the start and end position of the occurrence. These positions are given in lexical units.

If the file has the heading line `#I`, the end position of each occurrence is immediately followed by a newline. Otherwise, it is followed by a space and a sequence of characters. In REPLACE mode, that sequence corresponds to the transduction produced for the recognized sequence. In MERGE mode, it represents the recognized sequences into which the transductions have been inserted. In MERGE or REPLACE mode, this sequence is displayed in the concordance. If the transductions have been ignored, the contents of the occurrence is extracted from the text file.

10.6.2 The file concord.txt

The file `concord.txt` is a text file that represents a concordance. Each occurrence is encoded in a line that is composed of three character sequences separated by a tab, representing the left context, the occurrence (possibly modified by transductions) and the right context.

10.6.3 The file concord.html

The `concord.html` file is an `HTML` file that represents a concordance. This file is encoded in UTF-8.

The title of the page is the number of occurrences it describes. The lines of the concordance are encoded as lines where the occurrences are considered to be hypertext lines. The reference associated to each of these lines has the following form: `<a href="X Y Z">`. `X` and `Y` represent the start and end position of the occurrence in characters in the file `name_of_text.snt`. `Z` represents the number of the phrase in which this occurrence appears.

All spaces are encoded like indivisible spaces (` ` in HTML), which allows the preservation of the alignement of the occurrences even if one of them, the one that is at the beginning of the file, has a left context with spaces.

NOTE: if the concordance has been constructed with the parameter `glossanet`, the HTML file obtains the same structure, except for the links. In these concordances, the occurrences are real links pointing at the web server of the GlossaNet application. For more information on GlossaNet, consult the link on the Unitex web site (http://www-igm.univ-mlv.fr/~unitex).

Here an example of a file:

```
<html lang=en>¶
<head>¶
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">¶
   <title>6 matches</title>¶
</head>¶
<body>¶
<font face="Courier new" size=3>¶
MAÃTRE, L'<a href="104 109 2">AUTRE</a> COMM<br>¶
TRE COMME <a href="116 126 2">DOMESTIQUE</a><br>¶
- , Ãl'tait <a href="270 277 3">habitÃl'e</a> pa<br>¶
'UN COMME <a href="94 100 2">MAÃTRE</a>, L'<br>¶
un de les <a href="314 321 3">membres</a> le<br>¶
la maison <a href="158 165 3">portant</a> le<br>¶
</font>¶
</body>¶
</html>¶
```

Figure 10.2 shows the page that corresponds to the file below.

Figure 10.2: Example of a concordance

10.7 Dictionaries

The compression of the DELAF dictionaries done by the program `Compress` produces two files: a `.bin` file that represents the minimal automaton of the inflected forms of the dictionaries, and a `.inf` file that contains the compressed forms allowing the constructions of the dictionaries to be reconstructed from the inflected forms. This section describes the format of these two file types, as well as the format of the file `CHECK_DIC.TXT`, which contains the result of the verification of a dictionary.

10.7.1 The .bin files

A .bin file is a binary file that represents an automaton. The first 4 bytes of the file represent an entity that indicates the size of the file in bytes. The states of the automaton sare encoded in the following way:

- the two first bytes indicates if the state is final as well as the number of transitions that leave it. The highest bit is 0 if the state is final, 1 if not. The other 15 bits encode the number of transitions. Example: a non-final state with 17 transitions is encoed by the

  hexadecimal sequence 8011

- if the state is final, the three following bytes encode the index in the `.inf` file of the compressed form to be used to reconstruct the dictionary lines for this inflected form.

  Example: if the state refers to the compressed form with the index 25133, the corresponding hexadecimal sequence is `00622D`

- each leaving transition is then encoded in 5 bytes. The first 2 bytes encode the character that labels the transition, and the three following encode the byte position of the result state in the `.bin` file. The transitions of a state are encoded next to each other.

  Example: a transition that is labled with the letter `A` pointing at the state of which the description starts at byte 50106, is represented by the hexadecimal sequence `004100C3BA`.
  Exemple: une transition étiquetée par le caractère `A` pointant vers l'état dont la description débute au

By convention, the first state of the automaton is the initial state.

10.7.2 The .inf files

A `.inf` file is a text file that describes the compressed files that are associated to a `.bin` file. Here an example of a `.inf` file:

```
0000000006¶
_10\0\0\7.N¶
.PREP¶
_3.PREP¶
.PREP,_3.PREP¶
1-1.N+Hum:mp¶
3er 1.N+AN+Hum:fs¶
```

The first line of the file indicates the number of compressed forms that it contains. Each line can contain one or more compressed forms. If there are multiple forms, they are separated by commas. Each compressed form is made up of a sequence that allows to find a

canonical form again starting from an inflected form, followed by a sequence of grammatical, semantic and inflection codes that are associated to the entry.

The mode of the compression of the canonical form varies with the function of the inflectd form. If the two forms are identical, the compressed form summarizes the grammatical, semantic and inflectionary information like this:

```
.N+Hum:ms
```

If the forms are different, the compression program cuts up the two forms in units. These units can be a space, a hyphen, or a sequence of characters that contain neither a space nor a hyphen. This way of cutting up units allows to efficiently take into account the inflections of the composed words.

If the inflected and the canonical form do not carry the same number of units, the the program encodes the canonical form by the number of characters to remove from the inflected form followed by the characters to append. Thus, the first line of the file below corresponds to the line in the dictionary:

```
James Bond,007.N
```

Since the sequence `James Bond` contains three units and `007` only one, the canonical form is encoded with `_10\0\0\7`. The `_` character indicates that the two forms do not have the same number of units. The following number (here 10) indicates the number of characters to remove. The sequence `\0\0\7` indicates that the sequence `007` should be appended. The digits are preceeded by the `\` character so they will not be confused with the number of characters to remove.

Whenever the two forms have the same number of units, the units are compressed two by two. If the two units are composed with a space or a hyphen, the compressed form of the unit is the unit itself, like in the following line:

```
1-1.N+Hum:mp
```

This allows to maintain a certain visibility in the `.inf` file whenever the dictionary contains composed words.

Whenever at least one of the units is neither a space nor a hyphen, the compressed form is composed of a number of characters to remove followed by the sequence of characters to append. Thus, the dictionary line:

```
première partie,premier parti.N+AN+Hum:fs
```

is encoded by the line:

```
3er 1.N+AN+Hum:fs
```

The code `3er indicates that 3 characters are to be removed from the sequence \ve` and the characters `er` are to be appended to obtain `premier`. The `1` indicates that only one character needs to be removed from `partie` to obtain `parti`. The number `0` is used whenever it needs to be indicated that no letter should be removed.

10.7.3 The file CHECK_DIC.TXT

This file is produced by the dictionary verification program `CheckDic`. It is a text file that contains information about the analysed dictionary and has four parts.

The first part is the possibly empty list of all syntax errors found in the dictionary: missing of the inflected or the canonical form, the grammatical code, empty lines, etc. Each error is described by the number of line it concerns, a message describing the error, and the contents of the line. Here an example of a message:

```
Line 12451: no point found
jardin,N:ms
```

The second and third part display the list of grammatical codes and/or semantic and inflection codes respectively. In order to prevent coding errors, the program reports encodings that contain spaces, tabs, or non-ASCII characters. In addition to that, if a Greek dictionary contains the code `ADV` or the character `A` and the Greek `A` is used instead of the Latin `A`, the program reports the following warning:

```
ADV warning: 1 suspect char (1 non ASCII char): (0391 D V)
```

Non-ASCII characters are indicated by their hexadecimal character number. In the example below, the code `0301` represents the Greek `A`. The spaces are indicated by the sequence `SPACE`:

```
Km s warning: 1 suspect char (1 space): (K m SPACE s)
```

When the following dictionary is verified:

```
1,2 et 3!,.INTJ
abracadrabra,INTJ
saperlipopette,.INTJ
zut,. INTJ
```

the following file `CHECK_DIC.TXT` is obtained:

```
Line 1: unprotected comma in lemma¶
1,2 et 3!,.INTJ¶
Line 2: no point found¶
ah,INTJ¶
-----------------------------------¶
```

```
----  All chars used in forms  ----¶
-----------------------------------¶
  (0020)¶
! (0021)¶
, (002C)¶
1 (0031)¶
2 (0032)¶
3 (0033)¶
I (0049)¶
J (004A)¶
N (004E)¶
T (0054)¶
a (0061)¶
e (0065)¶
h (0068)¶
p (0070)¶
r (0072)¶
s (0073)¶
t (0074)¶
u (0075)¶
z (007A)¶
-------------------------------------------------------------¶
----    2 grammatical/semantic codes used in dictionary  ----¶
-------------------------------------------------------------¶
INTJ¶
 INTJ warning: 1 suspect char (1 space): (SPACE I N T J)¶
-----------------------------------------------------¶
----    0 inflectional code used in dictionary  -----¶
-----------------------------------------------------¶
```

10.8 Configuration files

10.8.1 The file Config

Whenever the user modifies his preferences for a given languages, these modifications are saved in a text file named 'Config' which can be found in the directory of the current language. The file has the following syntax:

```
TEXT FONT NAME=Courier new¶
TEXT FONT STYLE=0¶
TEXT FONT SIZE=11¶
CONCORDANCE FONT NAME=Courier new¶
CONCORDANCE FONT HTML SIZE=3¶
INPUT FONT NAME=Times New Roman¶
INPUT FONT STYLE=0¶
```

```
INPUT FONT SIZE=10¶
OUTPUT FONT NAME=Times New Roman¶
OUTPUT FONT STYLE=1¶
OUTPUT FONT SIZE=12¶
DATE=true¶
FILE NAME=true¶
PATH NAME=false¶
FRAME=true¶
RIGHT TO LEFT=false¶
BACKGROUND COLOR=16777215¶
FOREGROUND COLOR=0¶
AUXILIARY NODES COLOR=13487565¶
COMMENT NODES COLOR=16711680¶
SELECTED NODES COLOR=255¶
ANTIALIASING=false¶
HTML VIEWER=¶
```

The first three lines indicate the name, the style and the size of the font used to display texts, dictionaries, lexical units, sentences in text automata, etc.

The parameters `CONCORDANCE FONT NAME` and `CONCORDANCE FONT HTML SIZE` define the name, tha size and the font to use when displaying concordances in HTML. The size of the font has a value between 1 and 7.

The parameters `INPUT FONT ...` and `OUTPUT FONT ...` define the name, the style and the size of the fonts used for displaying the paths and the transductions of the graphs.

The following 10 parameters correspond to the parameters given in the headings of the graphs. Table 10.3 describes the correspondances.

Parameters in the `Config` file	Parameters in the `.grf` file
DATE	DDATE
FILE NAME	DFILE
PATH NAME	DDIR
FRAME	DFRAME
RIGHT TO LEFT	DRIG
BACKGROUND COLOR	BCOLOR
FOREGROUND COLOR	FCOLOR
AUXILIARY NODES COLOR	ACOLOR
COMMENT NODES COLOR	SCOLOR
SELECTED NODES COLOR	CCOLOR

Table 10.3: Meaning of the parameters

The parameter `ANTIALIASING` indicates whether the graphs as well as the sentence automata are displayed by default with the antialiasing effect.

The parameter `HTML VIEWER` indicates the name of the navigator to use for displaying the concordances. If no navigator name is defined, the the concordances are displayed in a Unitex window.

10.8.2 The file system_dic.def

The file `system_dic.def` is a text file that describes the list of system dictionaries that are applied by default. This file can be found in the directory of the current language. Each line corresponds to a name of a .bin file. The system dictionaries are in the system directory, and in that directory in the sub-directory `(current language)/Dela`. Here an example of the file:

```
delacf.bin¶
delaf.bin¶
```

10.8.3 The file user_dic.def

The file `user_dic.def` is a text file that describes the list of dictionaries the user has defined to apply by default. This file is in the directory of the current language and has the same format as the file `system_dic.def`. The dictionaries need to be in the sub-directory `(current language)/Dela` of the personal directory of the user.

10.8.4 The file user.cfg

Under Linux, Unitex expects the personal directory of the user to be called `unitex` and expects it to be in his root directory (`$HOME`). Under Windows, it is not always possible to associate a directory to a user per default. To compensate for that, Unitex creates a `.cfg` file for each user that contains the path to his personal directory. This file is saved under the name `(user login).cfg` in the sub-directory of the system `Unitex/Users`.

ATTENTION: THIS FILE IS NOT IN UNICODE AND THE PATH OF THE PERSONAL DIRECTORY IS NOT FOLLOWED BY A NEWLINE.

10.9 Various other files

For each text Unitex creates multiple files that contain information that are designed to be displayed in the graphical interface. This section describes these files.

10.9.1 The files dlf.n, dlc.n et err.n

These three files are text files that are stored in the text directory. They contain the number of lines of the files `dlf`, `dlc` and `err` respectively. These numbers are followed by a newline.

10.9.2 The file stat_dic.n

This file is a text file in the directory of the text. It has three lines that contain the number of lines of the files `dlf`, `dlc`, and `err`.

10.9.3 The file stats.n

This file is in the text directory and contains a line in the following form:

```
3949 sentence delimiters, 169394 (9428 diff) tokens, 73788 (9399) simple
 forms, 438 (10) digits¶
```

The numbers indicated are interpreted in the following way:

- `sentence delimiters`: number of sentence separators (`{S}`);
- `tokens`: total number of lexical units in the text. The number preceeding `diff` indicates the number of different units;
- `simple forms`: the total number of lexical units in the text that are composed of letters. The number in parentheses represents the number of different lexical units that are composed of letters;
- `digits`: the total number of digits used in the text. The number in parentheses indicates the number of different digits used (10 at the most).

10.9.4 The file concord.n

The file `concord.n` is a text file in the directory of the text. It contains information on the last search done on the text and looks like the following:

```
6 matches¶
6 recognized units¶
(0.004% of the text is covered)¶
```

The first line gives the number of found occurrences, and the second the name of units covered by these occurrences. The third line indicates the ratio between the covered units and the total number of units in the text.

Contents

Introduction **3**

1 Installation of Unitex **5**
1.1 The Java runtime environment 5
1.2 Installation on Windows 5
1.3 Installation on Linux 6
1.4 First Start 6
1.5 Adding languages 7
1.6 Deinstallation 7

2 Loading texts **9**
2.1 Selecting the language 9
2.2 Text formats 9
2.3 Opening a text 11
2.4 Text preprocessing 11
2.4.1 Normalisation of Separators 12
2.4.2 Phrase Detection 13
2.4.3 Normalization of non-ambigue forms 15
2.4.4 Splitting a text into lexical units 16
2.4.5 Applying dictionaries 17
2.4.6 Analysis of composite words in Norvegian 20

3 Dictionaries **21**
3.1 The DELA Dictionaries 21
3.1.1 The DELAF Format 21
3.1.2 The DELAS Format 24
3.1.3 Dictionary Contents 24
3.2 Verfication of the dictionary format 26
3.3 Sorting 28
3.4 Automatic flexion 29
3.5 Compression 31
3.6 Applying dictionaries 32
3.6.1 Priorities 32
3.6.2 Application rules for dictionaries 33

4 Search for regular expressions **35**
4.1 Definition . . . 35
4.2 Lexical units . . . 35
4.3 Patterns . . . 36
4.3.1 Special symbols . . . 36
4.3.2 References to the dictionaries . . . 36
4.3.3 Grammatical and semantic constraints . . . 37
4.3.4 Inflectional constraints . . . 37
4.3.5 Negation of a pattern . . . 38
4.4 Concatenation . . . 39
4.5 Union . . . 40
4.6 Kleene star . . . 40
4.7 Search . . . 41
4.7.1 Configuration of the search . . . 41
4.7.2 Presentation of the results . . . 42

5 Local Grammars **45**
5.1 The Local Grammar Formalism . . . 45
5.1.1 Algebraic Grammars . . . 45
5.1.2 Extended Algebraic Grammars . . . 46
5.2 Editing Graphs . . . 46
5.2.1 Import of Intex Graphs . . . 46
5.2.2 Creating a Graph . . . 47
5.2.3 Sub-Graphs . . . 49
5.2.4 Manipulating boxes . . . 50
5.2.5 Transducers . . . 51
5.2.6 Using Variables . . . 51
5.2.7 Copying Lists . . . 52
5.2.8 Special Symbols . . . 53
5.2.9 Toolbar Commands . . . 54
5.3 Display options . . . 55
5.3.1 Sorting the lines of a box . . . 55
5.3.2 Zoom . . . 55
5.3.3 Antialiasing . . . 56
5.3.4 Box alignment . . . 56
5.3.5 Display, Options and Colors . . . 57
5.4 Graphs outside of Unitex . . . 60
5.4.1 Inserting a graph into a document . . . 60
5.4.2 Printing a Graph . . . 61

6 Advanced use of graphs **63**
6.1 Types of graphs . . . 63
6.1.1 Inflection graphs . . . 63
6.1.2 Preprocessing graphs . . . 64
6.1.3 Graphs for normalizing the text automaton . . . 65

6.1.4 Syntactic graphs . . . 66
6.1.5 Model graphs . . . 66
6.2 Compilation of a grammar . . . 66
6.2.1 Compilation of a graph . . . 66
6.2.2 Approximation with a finite state transducer . . . 67
6.2.3 Constraints on grammars . . . 69
6.2.4 Error detection . . . 70
6.3 Rules for the application of transducers . . . 71
6.3.1 Insertion to the left of the matched pattern . . . 71
6.3.2 Application while progressing . . . 72
6.3.3 Priority of the leftmost match . . . 72
6.3.4 Priority of the longest match . . . 73
6.3.5 Transductions with variables . . . 73
6.4 Application of graphs to texts . . . 75
6.4.1 Configuration of the search . . . 76
6.4.2 Concordance . . . 77
6.4.3 Modification of the text . . . 77

7 Text automata 79
7.1 Presentation . . . 79
7.2 Construction . . . 80
7.2.1 Rules of construction of text automata . . . 81
7.2.2 Normalization of ambiguous forms . . . 82
7.2.3 Normalization of clitical pronouns in Portuguese . . . 82
7.2.4 Conservation of better paths . . . 84
7.3 Manipulation . . . 86
7.3.1 Displaying phrase automata . . . 86
7.3.2 Modify the text automaton . . . 87
7.3.3 Parametres of presentation . . . 88

8 Lexicon Grammar 89
8.1 The lexicon grammar tables . . . 89
8.2 Conversion of a table into graphs . . . 89
8.2.1 Principle of template graphs . . . 89
8.2.2 Format of the table . . . 90
8.2.3 The template graphs . . . 91
8.2.4 Automatic generation of graphs . . . 91

9 Use of external programs 95
9.1 Asc2Uni . . . 95
9.2 CheckDic . . . 97
9.3 Compress . . . 97
9.4 Concord . . . 97
9.5 Dico . . . 99
9.6 Extract . . . 99

9.7 Flatten . . . 100
9.8 Fst2Grf . . . 100
9.9 Fst2Txt . . . 100
9.10 Grf2Fst2 . . . 101
9.11 Inflect . . . 101
9.12 Locate . . . 101
9.13 MergeTextAutomaton . . . 102
9.14 Normalize . . . 102
9.15 PolyLex . . . 103
9.16 Reconstrucao . . . 103
9.17 Reg2Grf . . . 103
9.18 SortTxt . . . 104
9.19 Table2Grf . . . 104
9.20 TextAutomaton2Mft . . . 104
9.21 Tokenize . . . 104
9.22 Txt2Fst2 . . . 105
9.23 Uni2Asc . . . 106

10 File formats **107**
10.1 Unicode Little-Endian encoding . . . 107
10.2 Alphabet files . . . 108
10.2.1 Alphabet . . . 108
10.2.2 Sorted alphabet . . . 109
10.3 Graphs . . . 109
10.3.1 Format .grf . . . 109
10.3.2 Format .fst2 . . . 112
10.4 Texts . . . 114
10.4.1 .txt files . . . 114
10.4.2 .snt Files . . . 114
10.4.3 File text.cod . . . 114
10.4.4 The file tokens.txt . . . 114
10.4.5 The files tok_by_alph.txt and tok_by_freq.txt . . . 114
10.4.6 The file enter.pos . . . 115
10.5 Text Automaton . . . 115
10.5.1 The file text.fst2 . . . 115
10.5.2 The file cursentence.grf . . . 116
10.5.3 The file sentenceN.grf . . . 116
10.5.4 The file cursentence.txt . . . 116
10.6 Concordances . . . 116
10.6.1 The file concord.ind . . . 116
10.6.2 The file concord.txt . . . 117
10.6.3 The file concord.html . . . 117
10.7 Dictionaries . . . 118
10.7.1 The .bin files . . . 119
10.7.2 The .inf files . . . 119

10.7.3 The file CHECK_DIC.TXT 121
10.8 Configuration files 122
10.8.1 The file Config 122
10.8.2 The file system_dic.def 124
10.8.3 The file user_dic.def 124
10.8.4 The file user.cfg 124
10.9 Various other files 124
10.9.1 The files dlf.n, dlc.n et err.n 124
10.9.2 The file stat_dic.n 125
10.9.3 The file stats.n 125
10.9.4 The file concord.n 125

Index

[, 3
+, 8, 16
!, 6
#, 4, 6, 34
$, 19, 20
*, 8
-, 5
., 7
/, 19
:, 17
<CDIC>, 4
<DIC>, 4, 6
<E>, 4, 6, 8, 15, 32, 34
<MAJ>, 4, 6
<MIN>, 4, 6
<MOT>, 4
<NB>, 4, 6
<PRE>, 4, 6
<SDIC>, 4
<^>, 32
Flatten, 35
Fst2Grf, 55
Grf2Fst2, 34
L, 31
R, 31
Reconstrucao, 50
Uni2Asc, 14
\, 3
_, 20
{S}, 7

Algebraic Languages, 14
All matches, 9, 44
Antialiasing, 24, 27
Approximation of a grammar with a finite state transducer, 35
Automata
    finite state, 14
Automatic inflection, 31
Automaton
    acyclic, 48
    of the text, 5, 47
    text, 33
Axiom, 13

Box Alignement, 24
Boxes
    alignement, 24
    connecting, 16
    Creating, 15
    Deleting, 18
    Selection, 18
    sorting lines, 23
brackets, 8

Case
    seeRespect
        of lowercase/uppercase, 34
Case-sensitivity, 4
Clitics
    normalization, 50
Colors
    Configuration, 25
Comments
    in a graph, 16
Compilation of a graph, 34
Concatenation of regular expressions, 7
concatenation of regular expressions, 3
Concordance, 10, 45
Conservation of better paths, 52
Constraints on grammars, 37
Contexts
    concordance, 11, 45
    copy of a list, 21

Copy, 18, 20, 22
Copying Lists, 20
Creating a Box, 15
Cut, 22

Degree of ambiguity, 49
Derivation, 13
Dictionaries
    granularity, 49
    of the text, 4
    refer to, 4
Dictionnaries
    of the text, 47
    reference to, 34
Déplacer des groupes de mots, 41

Error detection in the graphs, 38
Errors in the graphs, 38
Exclusion of grammatical and semantic codes, 5
External Programs
    `Uni2Asc`, 14
External programs
    `Flatten`, 35
    `Fst2Grf`, 55
    `Grf2Fst2`, 34
    `Reconstrucao`, 50

Fichier
    `.fst2`, 34
File
    `.fst2`, 10, 54
    `.grf`, 10, 39, 55
    `.txt`, 45
    HTML, 11, 45

Grammars
    constraints, 37
    context-free, 13
    Extended Algebraic, 14
    for phrase boundary recognitions, 32
    Formalism, 13
    local, 34
    normalisation
        of non-ambiguous forms, 32
        of the text automaton, 33

Granularity of dictionaries, 49
Graph
    antialiasing, 24
    approximation with a finite state transducer, 35
    Box Alignment, 24
    Calling a Sub-Graph, 17
    comments in, 16
    compilation, 34
    connecting boxes, 16
    Creating a Box, 15
    Deleting Boxes, 18
    detection of errors, 38
    display, 23
    Display, Options and Colors, 25
    inflection, 31
    model, 34
    Printing, 29
    syntactic, 34
    types of, 31
    Variables in a, 19
    zoom, 23
Graphe
    antialiasing, 27
    including into a document, 28
    sauvegarde, 17
Graphs
    Intex, 14
Grid, 25

Import of Intex Graphs, 14
Including a graph into a document, 28
Infinite loops, 37
Inflectional constraints, 5

Kleene, *see* Kleene star
Kleene star, 3
Kleene star, 8

Lexical labels, 4, 49
Lexical units, 3
Longest matches, 9, 44
Lowercase
    seeRespect
        of lowercase/uppercase, 34

MERGE, 39, 44
Meta characters, 21
Meta-symbols, 4
Modification of the text, 45
Multiple Selection, 18
    copy-paste, 18

Negation, 6
non-terminal symbols, 13
Normalisation
    of ambiguous forms, 33
    of the text automaton, 33
Normalization
    of ambiguous forms, 50
    of clitics in Portuguese, 50
    of the text automaton, 50

Occurrences
    number of, 9, 44
Operator
    L, 31
    R, 31
    concatenation, 7
    disjunction, 8
    Kleene, 8
Options
    Configuration, 25

Paste, 18, 20
Paster, 22
Pattern, 4
Pixellisation, 24
Portuguese
    normalization of clitics, 50
Preferences, 27
Print
    a phrase automaton, 56
Printing
    a graph, 29
Priority
    of the leftmost match, 40
    of the longest match, 41

Rational Expressions, 14
Recursive Transition Networks, 14
Reference to dictionnaries, 34
References to the dictionionaries, 4
Regular expressions, 3
REPLACE, 39, 44
Respect
    des minuscules/majuscules, 33
    of lowercase/uppercase, 32, 34
    of spaces, 34
RTN, 14
Rules
    for transducer application, 39
    rewriting, 13

Search for patterns, 9, 44
Separator
    of phrases, 7
Shortest matches, 9, 44
Sorting
    lines of a box, 23
    of concordances, 11, 45
Space
    obligatory, 4
    prohibited, 4
State
    Final, 15
    Init, 15
Symbols
    non-terminal, 13
    special, 21
    terminal, 13
Syntax Diagrams, 14

Text
    automaton of the, 5
    modification, 45
    normalisation of the automaton, 33
    normalization of the automaton, 50
    preprocessing, 32
Toolbar, 22
Transducer, 14
    rules for application, 39
Transducers, 19
    with variables, 19
Transduction, 14, 26
    associated to a subgraph, 37
    with variables, 41

Types of graphs, 31

Underscore, 20, 41
Unicode, 14, 23
Union of ratinal expressions, 3
Union of regular expression, 8
Uppercase
    seeRespect
      of lowercase/uppercase, 34

Variable names, 20
Variables
    in graphs, 41
    within graphs, 19

Web browser, 11, 45
Words
    composed, 4
    simple, 4
    unknown, 6

Zoom, 23

Bibliography